CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

The Impact of Design Patterns on Quality Attributes in ML-Enabled Systems

A Multivocal Study of Component Models

Master's thesis in Computer science and engineering

Erik Eriksson
Joel Olausson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

The Impact of Design Patterns on Quality Attributes in ML-Enabled Systems

A Multivocal Study of Component Models

Erik Eriksson
Joel Olausson

UNIVERSITY OF GOTHENBURG

CHALMERS UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

The Impact of Design Patterns on Quality Attributes in ML-Enabled Systems
A Multivocal Study of Component Models
Erik Eriksson Joel Olausson

Supervisor: Vladislav Indykov, Department of Computer Science and Engineering
Co-supervisor: Daniel Strüber, Department of Computer Science and Engineering
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in $\LaTeX$
Gothenburg, Sweden 2024

The Impact of Design Patterns on Quality Attributes in ML-Enabled Systems
A Multivocal Study of Component Models
Erik Eriksson Joel Olausson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As machine learning is becoming a more common part of software systems, the need for new and improved architectural strategies due to the unique architectural challenges these systems introduce is becoming apparent. Right now, there is a lack of knowledge on how to build good architecture for such systems due to their nature of being rigid, vast, and dependent on data. This study analyses architectural design patterns in machine learning-enabled systems, and how they impact the quality of the system as a whole, to enable practitioners to make better architectural design decisions. The study was conducted by doing a multivocal literature review to extract component models from which architectural design patterns were derived. The connection to, and impact of, these patterns on the quality attributes of the system were then evaluated by interviewing experts. The result of this study is a set of 14 patterns, and the evaluated impact they have on the quality attributes of the system. These findings allow practitioners to choose design patterns based on the sought qualities for their system, making their architectural design decisions better.

Keywords: Software engineering, ML-enabled systems, software architecture, component models, design patterns, quality attributes.

Acknowledgements

We would like to thank our supervisors Vladislav Indykov and Daniel Strüber for all their guidance, our examiner Jeniffer Horkoff for valuable input, all of our interviewees for participating, Simon Andersson and Viktor Berggren for their comments, Oskar Giljegård for helping us with the pilot interview, and finally all the group rooms on Chalmers in which the work on this thesis was done[1].

Erik Eriksson, Joel Olausson, Gothenburg, June 2024

[1]https://docs.google.com/spreadsheets/group-room-ratings

Contents

List of Figures

List of Tables

Nomenclature

AI	Artificial Intelligence
DSAI	Division of Data Science and AI
GL	Grey Literature
IDSE	Division of Interaction Design and Software Engineering
ML	Machine Learning
MLR	Mulitvocal Literature Review
QA	Quality Attribute
RQ	Research Question
SLR	Systematic Literature Review
SWEBOK	Software Engineering Body of Knowledge
UML	Unified Modelling Language

1
Introduction

Artificial Intelligence (AI) has recently really stepped into the limelight. With this increase in popularity, more and more software systems include AI components, often based on Machine Learning (ML). ML code is only a fraction of the whole ML system compared to the large and complex infrastructure that supports it [1], which is why they introduce many unique architectural challenges when implemented into a software system. These challenges originate from aspects like the prominence of data handling and processing, drift, and an extended set of relevant quality attributes for ML systems, including model accuracy and data quality [2]. Further, developing ML models is often experimental and iterative, and ML systems themselves are often very rigid [3], clashing with the common agile way of working for the rest of the software system. Therefore, more modularity and evolvability for these types of systems are needed to increase development speed. Despite ML being a popular research topic, researching architecture of ML-enabled systems is often neglected. This means that many challenges still exist with no clear answer to them [2], [4], [5], even though some of these concerns have been around for a very long time [6]. Some of the most important challenges yet to be solved include a lack of architecture frameworks, processes, and evolution strategies for developing ML-enabled systems [4].

A step in the right direction could be to use *component models* as a more prominent tool for designing ML-enabled systems. A component model is a high-level description of how components of a system interact and can be efficiently used to visualise the large-scale architecture of software systems [7]. However, architecting ML-enabled systems by using component models does not directly solve many problems. If you do not know the consequences of your actions when designing the system with a component model, you may still face the same problems as those who do not use them. The key here is what Bass et al. write in their book "Software Architecture in Practice" [8]:

> "If we know that certain kinds of architectural decisions lead to certain quality attributes in a system, then we can make those decisions and rightly expect to be rewarded with the associated quality attributes."

To evaluate the architectural decisions made in the component models of ML-enabled systems, we will look into the *architectural design patterns* of them. A design pattern is a general solution to a reoccurring problem in software [9] and can describe both higher-level architecture and organization of code as well as lower-level design and implementation details [10]. In other words, all design patterns must solve some reoccurring problem. Architectural design patterns are design patterns within the software architecture design, and are therefore the subset of design patterns that are of most interest in this work. Henceforth in this thesis, the term *pattern* will be used interchangeably with architectural design pattern, while a general design pattern will always be referred to as *design pattern*.

1.1 Purpose

Discovering and defining architectural design patterns found in component models will enable higher levels of maintainability since it allows for documentation of them [11]. However, the main purpose of this thesis is to provide an impact analysis of patterns found in component models in terms of how they affect the quality attributes of the system, to allow practitioners to create better ML-enabled systems, faster. By connecting patterns that are used in component models with quality attribute impact, one can use those patterns when building a system and expect the associated quality attribute benefits [8]. This will make building ML-enabled systems faster, easier, and with less risk, since you can draw conclusions about the qualities of the system before it is implemented.

A secondary purpose is to create a base for further studies in the area of qualitative impact of patterns on the architecture of ML-enabled systems.

1.2 Research Questions

To fulfill the purpose of this thesis we will need to answer these three Research Questions (RQs).

RQ1: What common patterns can be derived from existing component models of ML-enabled software systems in literature?

RQ2: What quality attributes are associated with the identified patterns?

RQ3: What impact do the patterns of the component models have on the quality attributes associated with them?

To answer RQ1, a literature review will have to be conducted. To the best of our knowledge, no one has done a literature review to find component models of ML-enabled systems, or otherwise analysed component models of ML-enabled systems on a large scale before. There have been a few literature reviews on design patterns for AI-based systems, as will be described in section 3.2, but none about architectural design patterns that can be found in component models.

Similarly, based on our findings, no one has evaluated the usage of ML-related architectural design patterns on the quality of systems (i.e. using quality attributes), or in other ways looked at design patterns as a way to help build the architecture of ML-enabled systems. Answering RQ2 and RQ3 will therefore help with the architectural challenges of building ML-enabled systems.

1.3 Scope

The vast interest in ML-enabled systems [12] makes it infeasible, and way out of scope for this thesis, to find all common patterns from all component models of ML-enabled software systems. The aim of RQ1 is to get an understanding of the current state of research and practice when it comes to the usage of component models, and the patterns within them, for ML-enabled software systems. Answering it will provide insight into how the architectures of these systems are currently designed — what reoccurring problems that exist have general solutions that can be expressed as patterns in component models. Therefore, the answer to RQ1 will not contain every pattern to exist in any component model. The added benefit of going from deriving a snapshot of component model patterns to an exhaustive list of them is questionable since new component models with potentially new patterns are being created for ML-enabled systems as you are reading this, meaning that the fact that it is exhaustive loses its meaning immediately.

Now that we have established that an exhaustive search for component models and patterns is not the aim of RQ1, nor within the scope of this thesis, it is still important to highlight that it is crucial to have as large of a sample of component models as possible to extract patterns from. This is because a larger amount of component models give more opportunities to derive common patterns from them. Too few, and you may run into problems with generalisability, as well as issues of evaluating if a pattern is a common solution to a reoccurring problem or just an implementation-specific solution. For these reasons, we will consider as many component models as we believe we have time for during this thesis. However, we will only consider component models in the form of figures. A component model may not necessarily be presented visually, but will in most cases take an unnecessary amount of time to identify and comprehend otherwise.

For the purposes of this thesis, we will only consider a pattern as such if it can be found in at least two component models. This may reduce the number of patterns found but will ensure that the patterns are reoccurring — a prerequisite for it to be a pattern in the first place. Furthermore, this will save time that would otherwise have been spent trying to draw the line on what exactly is a reoccurring problem or just an implementation-specific solution.

2

Background

This chapter will introduce as well as define important concepts that are relevant for the rest of the thesis. Doing so will motivate some higher-level decisions that were made: Which type of systematic literature review is selected for this study, which definition of grey literature, and which quality model is used. The chapter concludes by concretising exactly what is meant by a component model in this thesis.

2.1 Literature Reviews

Conducting a Systematic Literature Review (SLR) is a technique commonly used for attempting to find all information about a certain topic [13]. The way SLRs are performed makes them fair and of high scientific value, in large part because they are transparent and reproducible, with as little bias as possible. A set of well-established guidelines exist for performing SLRs in software engineering, created by Kitchenham et al. [13].

Similar to these are Multivocal Literature Reviews (MLRs), a type of SLR where grey literature is included along with white literature, which gives the benefits of including sources of both researchers and practitioners [14]. In a quickly growing area of research, including grey literature in a review can be very beneficial [15], [16]. It also has the advantage of bringing research and practice closer together, especially important in areas with lots of practitioner interest or lack of corroboration between research and practice. Garousi et al. have created a set of guidelines for conducting MLRs in software engineering which can be used as variation points to Kitchenham et al.'s guidelines for conducting SLRs [14]. In these guidelines, a list of seven questions is presented. If at least one of these questions is answered with "yes" it indicates that grey literature should be included in the literature review.

The selected type of literature review for this study is an MLR, because of the benefits listed above. This field is rapidly growing [12] and with a lot of practitioner interest. Furthermore, three of the seven questions mentioned earlier for determining the inclusion of grey literature by Garousi et al. [14] were answered "yes". Similar research, which will be presented in section 3.2, has included grey literature to great success, which further highlights the value including grey literature can bring.

2.2 Grey Literature

Defining what Grey Literature (GL) is, is not trivial. The most widely accepted definition of grey literature is, or at least was, the Luxembourg definition [17], [18]. Since then, an updated Prague definition has been established that adjusts the definition in regards to publishing on the internet [18]. Both these definitions can be seen in Table 2.1, showing how the Prague definition adds intellectual property and quality constraints to the Luxembourg definition. In their guidelines for conducting MLRs in software engineering, Garousi et al. also highlight the existence of multiple definitions of GL [14] and imply to use an adapted version of a model by Adams et al. [19]. Adams et al.'s model is built on tiers of outlet control and credibility. However, Kitchenham et al. point out that these tiered models are "largely defined by example, which is a weak method of definition" [20], to which we agree. They instead suggest using the Prague definition for defining GL because it imposes GL to be *collected and preserved*, making the systematic review reproducible. Furthermore, they address that Kitchenham et al.'s guidelines for conducting systematic reviews [13] do not rule out GL conforming to the Prague definition, making Garousi et al.'s guidelines for conducting MLRs mostly obsolete.

Luxembourg definition	Prague definition
"information produced on all levels of government, academia, business and industry in electronic and print formats not controlled by commercial publishing, i.e. where publishing is not the primary activity of the producing body."	"Grey literature stands for manifold document types produced on all levels of government, academics, business and industry in print and electronic formats that are protected by intellectual property rights, of sufficient quality to be collected and preserved by library holdings or institutional repositories, but not controlled by commercial publishers i.e. where publishing is not the primary activity of the producing body."

Table 2.1: The Luxembourg and Prague definitions of GL [18], [20].

The problem with using the Prague definition is that it, of course, will leave out many potentially valuable sources because they do not conform to it. This goes against the purpose of our literature review, to determine which patterns are used in existing component models identified in literature, which is best solved by looking at both research and practice. The point of including grey literature to begin with is to get a snapshot of practitioners' use of component models. Therefore, we will sacrifice complete reproducibility for a definition of GL that makes more sense for the purpose of the literature review.

Henceforth in this thesis, GL will use the following definition, given by Garousi et al. in a more recent publication [21]:

> *Grey literature in SE can be defined is* [sic] *any material about SE that is not formally peer-reviewed nor formally published.*

This means that for example blogs may be used as grey literature sources or other sources with lower credibility. In our case, this is deemed acceptable. Each source must not have very high credibility since the patterns extracted will not be directly from an individual source, but rather from the component models within multiple sources. Therefore, the credibility of the sources will have a negligible effect on the quality of the answers to the research questions.

2.3 Quality Model

As we strive to create high-quality software systems “comprehensive specification and evaluation of the quality of software [...] is a key factor” [22]. Therefore, in 1991 the International Standards for Organisation (ISO) set a standard, ISO9126, for Software engineering product quality which since has been revised several times. The newest version (ISO25010:2023) [22] defines a quality model containing different quality characteristics to clearly outline where software systems can falter in quality. As quality attribute and quality characteristics are synonyms [23] we chose to use Quality Attribute (QA), even if ISO uses quality characteristics, since we feel it is a more commonly used term.

Since machine learning systems are uncertain systems that rely heavily on data, the quality of such systems cannot be exhaustively characterised by existing quality models for traditional software [24]. Therefore, ISO published ISO25059 [25], a new quality model for AI systems, which includes more of the AI/ML specific quality attributes needed to evaluate AI/ML-enabled systems. This model also removes some of the quality attributes of ISO25010 even though they could be useful for some systems. For this reason, Indykov et al. have done a systematic literature review to find which quality attributes are mentioned in literature about ML-enabled systems, and based on the results created their own quality model [24]. This quality model can be seen in Figure 2.1.

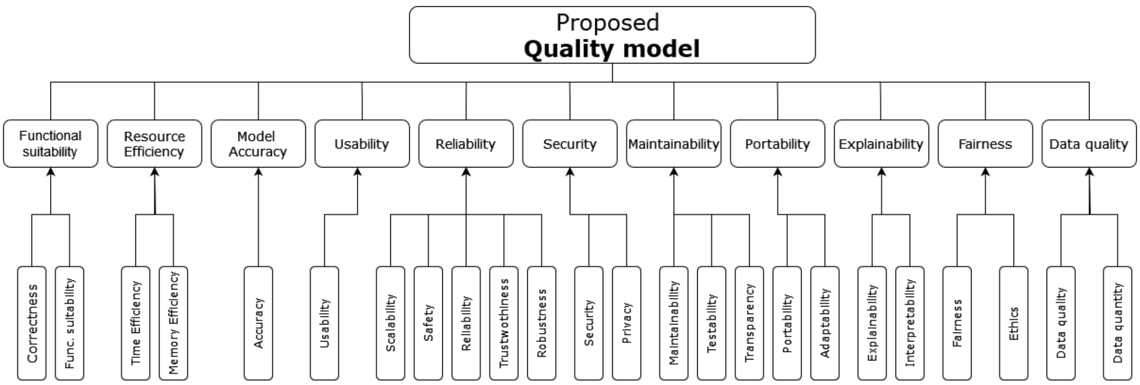

Figure 2.1: The quality model proposed in Indykov et. al’s [24] literature review.

As this model is constructed from what qualities are discussed in literature, one could conclude that this model could also be missing some quality attributes. Indykov et al. have seen this flaw as well and compared their model to the ISO25059 and

ISO25010 to see if they contain something that is missing in their model, which can be seen in Table 2.2.

Proposed Quality Model	**ISO 25059 (for AI systems)**	**ISO 25010 (for general software)**
Func. Suitability	Func. Correctness	Func. Suitability
Resource Efficiency	-	Perf. Efficiency
Usability	User Controlability	Usability
Reliability	Robustness	Reliability
Security	-	Security
Maintainability	Intervenability	Maintainability
Portability	Functional Adaptability	Portability
Explainability	Transparency	-
Model Accuracy	-	-
Fairness	Ethical	-
Data Quality	-	-
-	-	Compatibility

Table 2.2: Comparison of quality models, taken from Indykov et al. [24].

Since the model proposed by Indykov et al. contains all the quality attributes from ISO25059 and most from ISO25010, it is the model we have selected to use. We will, however, add compatibility to it since it could be a useful quality attribute and is included in the standard for general software. Even though compatibility might not have appeared when conducting the systematic literature review by Indykov et al., it does not mean that patterns do not impact that quality attribute. Therefore, it is valuable to include it in the quality model used in this work.

2.3.1 Quality Attribute Definitions

Knowing which quality attributes are used is one thing, knowing what they represent is another. Different people can interpret the QAs in different ways, which is why it is important to have a clear definition for them so that everyone is on the same page. Table 2.3 shows all quality attributes included in the quality model that is used in this thesis, together with their definitions. The definitions are taken from Indykov et al. [24] which, in a nutshell, are summarised versions of the definitions from the standards. This is with the exception of compatibility, which is taken from ISO 25010.

2.3.2 Functional Suitability Relevancy

The quality attribute *functional suitability* is directly tied to the functional requirements of the system. Even though it is a very important quality attribute, it cannot be taken into account in this work, since when studying the connection of general patterns to general quality changes, the implementation-specific quality changes will not translate to other systems by default. Functional suitability is still a part of

Quality Attribute	Definition
Functional Suitability	The degree to which a system provides functions that meet stated and implied needs.
Resource Efficiency	The degree to which a system fulfils a given functionality within an existing amount of hardware capacities.
Usability	The degree to which a system can be used by end users to achieve specified goals.
Reliability	The degree to which a system performs specified functions under specified conditions.
Security	The degree to which a system protects information and data.
Maintainability	The degree to which a system can be modified and supported by developers and maintainers to achieve specified goals.
Portability	The degree of effectiveness with which a system can be transferred from one hardware or software basis to another.
Explainability	The degree to which the behavior of the ML model and its output can be explained to humans.
Model Accuracy	The degree to which an ML model performs and analyzes the contextual environment.
Fairness	The degree to which a system can detect and prevent an algorithmic bias created by the ML model.
Data Quality	The degree of integrity, reliability, and sufficiency of data for model training, testing, and validation, and reliability of the related data sources.
Compatibility	The degree to which the system exchanges information; perform its required functions, while sharing the same hardware or software environment.

Table 2.3: Table of the quality attribute definitions from the quality model used in this study.

the quality model used, but will therefore not be a part of the answers to RQ2 and RQ3.

2.4 Component Models

According to the United States Department of Defense “A Component Model describes the hierarchy of functional components, their responsibilities, static relationships, and the way components collaborate to deliver required functionality” in software [7]. A component model is something unique to a system since it depicts how the specific components of that system interact, and thus cannot be used directly for other systems.

In the Unified Modeling Language (UML), component diagrams are a similar concept to component models. However, in contrast to them, component models can be illustrated more freely and on different levels of abstraction. In fact, component models do not have a standard, which may be one of the main reasons “component models” are not a widely used concept in software development. As a consequence of having no standard, component models can be depicted in an indeterminable number of ways. They can even be non-visual, for example in the form of tables or written in plain text; as long as they describe the hierarchy of components, etc., as [7] puts it, they can be classified as component models as well. This is a problem, since in a systematic literature review for finding component models, where you cannot be precise in describing what you are searching for, you lose some of its academic value because of reproducibility issues.

To combat this issue, two restrictions are added. First, while a component model may be non-visual, for the rest of this thesis only visual component models (in the form of figures) will be considered. This idea was first presented in section 1.3 and, as discussed there, will not only help with reproducibility, but it will also save great amounts of time during the literature review process. Second, while the United States Department of Defense’s definition of a component model is solid, it can also be overwhelming. For the literature review, the definition was simplified to “A component model describes the relationship between architectural components”. This simplification does not take anything away from the previous definition, since a relationship between architectural components implies collaboration.

3

Related Work

This chapter will present related research in three main areas: Architecture of ML-enabled systems, design patterns, and quality attributes and their connection to design patterns.

3.1 Software Architecture of ML-Enabled Systems

As software systems became larger and more complex, the need for good structure when designing software arose [26]. To fulfill this need Perry and Wolf published the first paper on software architecture in 1995, accurately named “Foundations for the Study of Software Architecture” [27]. This opened up the field of software architecture and allowed others to continue the work within the field to, for example, analyse common architectural styles [28], a precursor to architectural patterns. When the internet was introduced and systems became more connected, a new focus on the systems' non-functional qualities and how to achieve them arose [29].

Now, when more software systems start to integrate ML, we will need to reconsider how important data and algorithms are [30] since they are fundamental building blocks of ML-based systems [4], [31] but “don't appear prominently in software architecture literature” [30]. Therefore, there is a need for new techniques and architectures to be found as the old techniques do not take these factors into account. To do this, we can take a look at the relevant non-functional qualities (quality attributes) for ML-enabled systems (section 2.3) compared to regular software systems because the first step in finding good architectural tactics is to know what they need to focus on.

3.2 Architectural Design Patterns

Design patterns is not a new area of research in software engineering. On the contrary, it is a very well-researched area [32], with many systematic literature reviews on the subject. Within AI specifically, design patterns for responsible AI (i.e. the ability of AI to behave ethically and responsibly) [33], and multi-agent systems (i.e. a subfield of AI involving the interaction of intelligent agents) have been researched [34].

As the name implies, architectural design patterns are a subset of all design patterns that specifically solve architectural problems. There have been studies on architectural design patterns within more niche areas of software engineering like Internet of Things [35] and microservices [36], [37]. However, as far as we know, only two systematic reviews that include architectural design patterns for AI or ML have been conducted, which will be summarised in the following paragraphs.

As recently as a few years ago at the time of writing this, Washizaki et al. conducted a multivocal literature review for finding patterns for ML architecture and design, which they claimed was the first of its kind [38]. In total, they found 33 patterns and anti-patterns, though only two of them are described in detail. Later, Washizaki et al. continued the work on these patterns and narrowed them down to 15, removing those that were vaguely defined or had questionable usefulness [39]. The identified patterns are either very high-level architectural patterns, such as “Distinguish Business Logic from ML Models”, or more design-oriented, and therefore not about the high-level architecture.

In 2023, Heiland et al. conducted an MLR with a similar purpose — identifying design patterns for AI systems [40]. They identified 70 patterns, 36 of which they classified as “traditional patterns”, i.e. patterns that can be found in non-AI contexts but have been adapted to an AI context. The 34 other patterns were new. All patterns were categorised into different groups, for example, *deployment*, *implementation*, or *security and safety*. The category with the most patterns was *architecture* with 25, which the authors say adds to the current understanding of the importance of architecture and its reuse. It is not always clear if these architectural design patterns (or any of the other patterns) are for AI or ML only, or applicable for both.

3.3 Quality Attributes’ Connection to Design Patterns

It is established that using design patterns correctly makes the software better [11], [41]. Better programs mean an increase in quality that can be measured via quality attributes. It is therefore not unreasonable to assume that patterns have a direct impact on the quality attributes for a given system. This line of reasoning is not something new [41], many previous studies have tried to find the connection between design patterns and QAs [11], [37], [39], [42], [43].

However, evaluating quality attributes’ connection to patterns is a hard task, as stated by Mayvan et al. in a systematic mapping study on design patterns [32]. The authors found that evaluating QAs has been performed either through quantitative code metrics or through expert opinion.

Wedyan and Abufakher report results that are often inconclusive or contradicting in their SLR on the impact of design patterns on software quality, though with an indication that design patterns are a net positive for quality [11]. Maintainability is the QA that is evaluated the most, though not many agreements can be made whether or not certain patterns positively or negatively affect it. They state that a possible reason for this is the fact that there are lots of confounding factors outside of the pattern itself, e.g., programming language, program size, and quality of documentation. Results do agree on the importance of the aforementioned documentation, though — documenting design pattern instances greatly increases maintainability.

Some studies have, despite its challenges, attempted to answer the question of QAs' connection to design patterns. A study on microservice patterns found a different distribution for which quality attributes were associated with the microservice patterns compared to what Wedyan and Abufakher found [42]. Here, availability was the most common QA the patterns were associated with, which, given the area of research, is perhaps not surprising. Their method for identifying the QAs each architectural pattern addresses was to read literature connected to it in-depth. They also discussed the topic with experts and practitioners. Another paper used hypothesis testing to conclude which kinds of patterns impact software quality more positively than others [43]. To find which QAs were influenced by design patterns, they listed all QAs and used "exploration" as a means to find which were impacted by the design patterns. The only hypothesis (out of three) that was statistically significant was that structural design patterns impact software quality more positively than creational ones. A third paper, this time written by Valdivia et al., conducted a multivocal literature review to find patterns in microservices, which they then identified connections to quality attributes by reading the sources describing the patterns[1] [37]. If this did not provide enough information, they consulted another researcher to get someone else's opinion. These studies show that answering how the qualities of a system are connected to design patterns is viable.

[1]Valdivia et al.'s study ([37]) is coincidentally very related to this study because of the many similarities between this study and theirs. The two main differences are that this research is on ML-enabled systems instead of microservices, and the added impact evaluation of the quality attributes (RQ3) in this study, which was not included in Valdivia et al.'s study.

4 Methodology

The study was conducted using the design science research methodology, following Knauss' guidelines [44]. This is an advantageous way of working for this study. However, some deviations were made due to the nature of the thesis's goals. Below is an explanation of how design science was applied, together with the aforementioned deviations and the motivations behind them.

In design science, an artefact is developed. In this study, the artefact was defined to be the list of patterns that can be found in component models together with how they affect quality attributes. Building such an artefact will fulfil the purpose of this thesis. The artefact can be effectively used by practitioners to build component models that when implemented will make the system attain wanted qualities. For example, if the requirements for a system state the importance of maintainability, using patterns from the artefact that increase maintainability will be advantageous. Likewise, patterns that decrease certain quality attributes can be avoided if reaching some high level of these QAs is necessary for the system according to its requirements.

The three stated research questions do not perfectly follow the regulative cycle, where one is related to the problem, one is about the potential solutions, and one on how good the potential solutions are to the problem [44]. RQ1 does not answer the whole problem, i.e. the problem about constructing better architecture for ML-enabled software systems, since that would be an immense task, way out of scope for this thesis. However, it does relate to the problem that no one knows how to maximise the utilisation of component models as a tool for building ML-enabled systems. In this work, patterns are seen as a means to understand component models, and thereby also the architecture of ML-enabled systems, better. Therefore, RQ1 is still very much related to the problem, though it does not follow the usual format for design science research questions, which is answering what problems exist. As for the other RQs, RQ2 is about the potential solutions since its answer will give insight into how the patterns can be used to maximise the utilisation of component models to create better architecture for ML-enabled systems, and similarly, RQ3 does evaluate the potential solutions.

As per Knauss' guidelines [44], the whole process was divided into three cycles. These cycles are recommended to be regulative, meaning that the problem, solution, and evaluation are iteratively examined, where you focus on one RQ in each cycle but go back and forth between all of them to work towards all research questions in all cycles. However, only the last two cycles were regulative. Because of the nature of RQ1, not being about what problems exist, it is rather seen as a stepping-stone to build RQ2 and RQ3, the potential solutions and their evaluation, on top of it. Moreover, because a systematic literature review was used to answer RQ1, it makes most sense to finish the literature review before starting RQ2 and RQ3, since changing the literature review because of progress in RQ2 or RQ3 will introduce threats to the validity of the review. Therefore, the first cycle focused purely on RQ1.

The rest of this chapter is divided as follows. First, how the multivocal literature review was conducted to find component models of ML-enabled systems. Second, how the patterns were extracted from the component models found in the literature review. These sections together describe how RQ1 was answered. Finally, how interviews were planned, conducted, and extracted on data to answer RQ2 and RQ3.

4.1 Multivocal Literature Review

A multivocal literature review was conducted to systematically elicit data used to answer RQ1. We followed guidelines by Garousi et al. for conducting MLRs in software engineering [14]. As suggested in these guidelines, Kitchenham and Charters' guidelines for conducting SLRs in software engineering [13] were used, but with the former as variation points.

For the MLR, a review protocol was developed and tested with a pilot search. The review protocol that was developed can be seen in appendix C. The rest of this section will go into detail about how the pilot search was performed and the lessons learned from it, before the final review protocol is presented. Finally, the search process with how the sources were filtered in different phases will be described.

4.1.1 Pilot Search

Piloting a research protocol is essential to, for example, refine the study selection criteria or data extraction strategies [13]. In our case, it was also a critical step in perfecting the search strings and determining which databases for searching grey literature would be the most valuable.

The pilot search was performed by searching a given database and applying the review protocol to the first ten search results. Afterwards, it was discussed if the extracted data fit the expectations of the authors and if the selection criteria or search strings could be improved. This strategy was performed at least once for every database, but often more than once, for example when different search strings were tried. The measure for how successful the search was, was determined by the number of sources that fit our defined selection criteria (see section 4.1.3), which, in essence, equals the number of unique component models that were extracted.

Originally, the Google search engine was planned to be included as a grey literature database, along with Google Image search and Google Patents. However, the pilot search clearly established that it was redundant since the results were subsumed by Google Image search.

Many different search strings for grey literature were piloted. For Google Image search, an important consideration that must be taken into account is the limit of 32 words for a single query. However, this did not matter since having a too complex string seemed to dilute the results anyway. After trying different strings, it became clear that small variations of the search string yielded similar search results. Therefore, a short and simple search string for Google Image search was decided to be optimal.

4.1.2 Search Strategy

The search was performed in four databases for white and two for grey literature. For white literature, the databases IEEE Xplore Digital Library, Wiley Online Library, ACM Digital Library, and SpringerLink were used. These were selected to get a wide coverage of literature within the software engineering field. Grey literature was searched through the Google Images search engine as well as Google Patents.

The search strings were derived by assembling a list of synonyms and similar phrases to "component model" and "machine learning" respectively. The Software Engineering Body of Knowledge (SWEBOK) [10] was used to aid in the former.

For white literature, the search string was the following:

```
("component model*" OR "architectural description language*"
OR "component diagram*" OR "class responsibility collaborator*" OR
"structure chart*" OR "component based design*")
AND
("ml-system" OR "ml-enabled system" OR "machine learning" OR
"deep learning" OR "reinforcement learning" OR
"unsupervised learning" OR "supervised learning" OR "neural network")
AND
NOT "artificial intelligence"
```

The wildcard operator, marked by asterisks, specifies any number of unknown characters, which in this case is used to include inflections of some of the terms. This was deemed not needed for the machine learning terms.

There has been a huge rise in interest in artificial intelligence within the last decade, seen not least by the number of published publications on the matter [12]. Machine learning is a form of AI, which means that ML has followed a similar trend. To remove the most obsolete machine learning systems, all searches were filtered from 2014–2024. This should give good chances that the source is still relevant, even in this rapidly growing field.

A side effect of the trend on AI is that a large part of the research is not relevant when looking for machine learning systems only. Papers about artificial intelligence tend to be too broad even though they might include machine learning as well. For these reasons, papers that contained "artificial intelligence" were excluded. After the removal of "artificial intelligence", the amount of search results was roughly halved. Since the amount of sources before this was very high, this was a way to make the study feasible within the given time frame.

Since Wiley and SpringerLink are both databases that are not computer science-specific, the pilot search showed a lot of sources focused on how to apply machine learning in different fields. Due to this both of these databases were filtered on the area of computer science.

The search string for Google Images had to be modified since there is a limit of 32 words for a query. Different search strings were tested to find the one that gave the most promising results after a pilot search was performed, see section 4.1.1. The selected search string for Google Images was the following:

```
"machine learning" AND ("component model" OR "component diagram")
```

The thought behind it is that Google's algorithms will take into account synonyms and similar phrases automatically, resulting in a simple string for optimal results.

What was not known at the time of searching, is that Google Search and Google Image Search do not support parentheses in queries. This means that the de facto search string for Google Images was:

```
"machine learning" AND "component model" OR "component diagram"
```

While this oversight certainly may have decreased the quality of the search string, it will not have affected the outcomes of the literature review significantly. Some sources that only include "component diagram" and not "machine learning" may have been evaluated, however, these would not have passed the study selection criteria. This problem will be further discussed in section 6.5.1.1.

When searching grey literature in Google Images, a private browsing mode was utilised. This is because Google tailors the search results depending on your earlier activity [45]. A private browsing mode prevents this since it has no earlier activity to go off of [46].

Finally, the following string was used for Google Patents:

```
("ml-system" OR "ml-enabled system" OR "machine learning" OR
"deep learning" OR "reinforcement learning" OR
"unsupervised learning" OR "supervised learning" OR "neural network")
AND
NOT "artificial intelligence"
```

From piloting this search string and comparing it with the one used for white literature it was deemed that the former had higher potential. A theory for the reason behind this is that patents about machine learning very often already include architecture in the form of component models, making that part of the string unnecessary to include, to the point where it dilutes the search results.

4.1.3 Study Selection Criteria and Procedures

The following four inclusion criteria were applied, where each one had to be true for a source to be included:

- The system the source mentions is a software system. (1)
- The system the source mentions is a machine learning system. (2)
- The focus of the source is on the system itself. (3)
- The source contains one or more figures showing the relationship between architectural components. (4)

For the focus of the source to be on the system itself (inclusion criteria 3), either the whole source must revolve around a single system or the source must describe the system with great nuance. This is to not include sources that contain an ML-enabled system but only as a small part of the source and therefore does not take the time to explain it fully, or uses it only as a small example of something larger.

The exclusion criteria partially differed between white and grey literature because of their nature, though the first three are common between them. For white literature's exclusion criteria, the following were applied, where the source was excluded if it fulfilled any of the following:

- The source was not written in English.
- The source was a duplicate.
- The source was published before 2014.
- The source was not available in full text.
- The source was not a single chapter or paper.

For grey literature, the following were applied:

- The source was not written in English.
- The source was a duplicate.
- The source was published before 2014.
- The source was video or audio.
- The source contained component models or equivalent only as an example of what they are or how they are used.

The stopping criteria for grey literature was selected to be effort-bound to 100 sources for each database. One hundred was arbitrarily chosen and puts a lot of faith in Google's search ranking[1] to give the most relevant results first. However, the alternative stopping criteria, as suggested by [14], were judged as infeasible: Going through over 100,000 results is unreasonable, thereby excluding evidence exhaustion,

[1]https://www.google.com/search/how-search-works/ranking-results/

and there is no guarantee that theoretical saturation can be reached, that is, there might be thousands of component models if you search long enough.

The criteria were applied in the following procedures. Each source was assessed by one of the authors. If some criteria was difficult to assess it would be discussed with the other author. If disagreements arose, a third opinion would be sought.

4.1.4 Data Extraction Strategy

All identified figures of component models would be copied and saved with an identification number that traces back to the source and put into a data extraction form. In this form, personal notes were written which included a summary of the component model. This meant that only one of the authors had to read the source thoroughly (thoroughly enough to understand the component model) while the other author could later quickly read the summary to familiarise oneself with the source without having to spend much time on it. Columns were added for additional information about the system, like the type of machine learning that was present (e.g., reinforcement learning, neural networks, deep learning). Each component model was also categorised into which type of system it depicts: Either a whole system, a part of a system, or a pipeline.

4.1.5 Literature Review Process

The search for white and grey literature was done on the 1st of February 2024. All sources were listed in a document with titles and links so that sources that appeared multiple times could be removed. Out of 562 white literature sources, 8 were duplicates (and were removed from one of the databases it was included in arbitrarily). Further, 7 white literature sources were not available in full text and 9 were not a single chapter or paper and were therefore not considered because of the exclusion criteria.

The distribution of all (non-duplicate) sources can be seen in Figure 4.1. The two grey literature databases (indicated by grey slices) make up roughly one fourth of all sources, or 26.6%.

The study selection criteria were applied in two phases. In the first phase, exclusion criteria would be checked, since they do not require much effort and therefore could be applied in a quite short time frame. Then, inclusion criteria 4 would be applied, because, similar to the exclusion criteria, it could be done in a quick manner and without having to understand the source. After this first phase, 80 sources were still in consideration. In the second phase, the rest of the inclusion criteria were checked (criteria 1, 2, and 3). To do this, one could not only look at the figures anymore — more of the source had to be read and understood. Figure text, data commentary, abstract, and sometimes more of the source were read to gain insight whether it fulfilled the rest of the inclusion criteria. If doubts arose, it would be immediately discussed with the other author to minimise the risk of excluding a source that fulfilled the inclusion criteria. After all study selection criteria had been

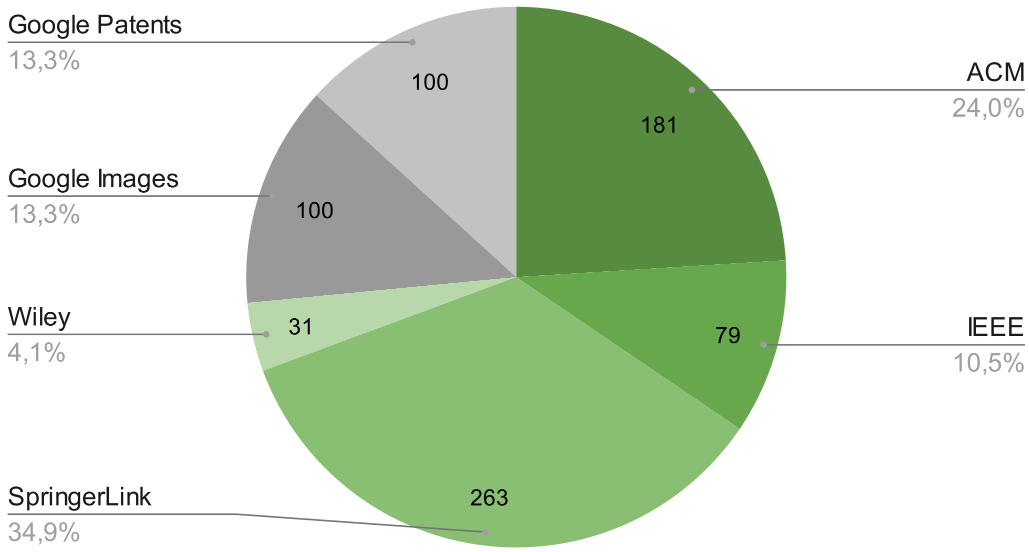

Figure 4.1: Pie chart showing the distribution of all sources in the six different databases searched. Green slices refer to white literature databases while grey slices refer to grey literature.

applied there were 49 sources left, from which an equal number of component models could be extracted.

The distribution for these 49 sources can be seen in Figure 4.2. Looking at the green slices, they do not tell us much other than some databases had more component models (IEEE and Wiley) than others (ACM and SpringerLink) relative to the number of sources that appeared in the search, comparing with Figure 4.1. What is perhaps more interesting is that the share of grey literature sources went from 26.6%, before the selection criteria had been applied, to 36.7% after. This increase is quite large, and almost all of it is from Google Images. This could be a sign of the value grey literature can bring to a literature review in software engineering — at least in this context. It also suggests that it could have been worthwhile to go through more grey literature sources than the 100 that were considered. Section 6.5.1.1 will further discuss this matter.

A summary of the number of sources for each database after each filtering step in the literature review process can be seen in Table 4.1. Out of the total 754 unique sources of white and grey literature, 49 component models were extracted, giving a source inclusion rate of about 6.5%.

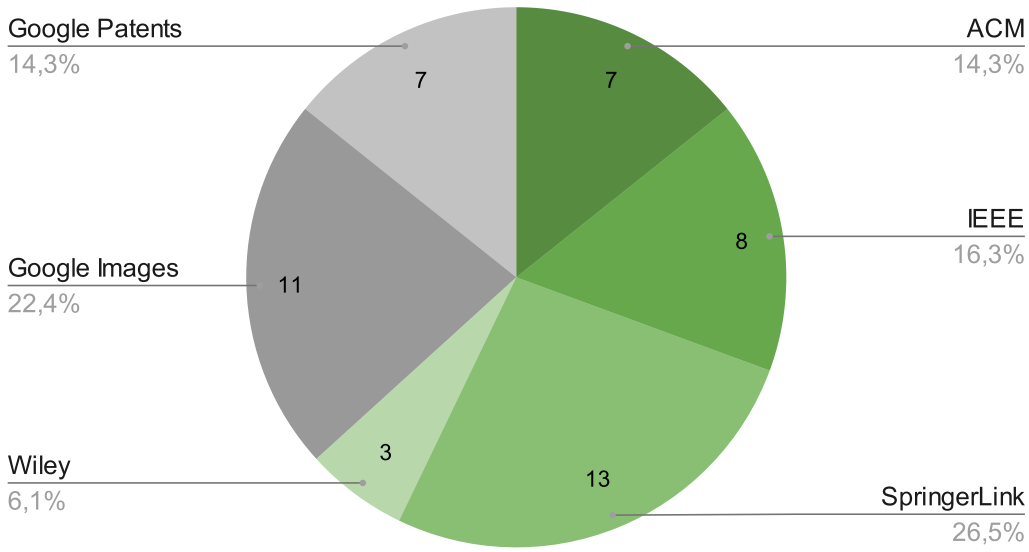

Figure 4.2: Pie chart showing the distribution of the sources that fulfilled all selection criteria for the six different databases searched. Green slices refer to white literature databases while grey slices refer to grey literature.

Step	ACM	IEEE Xplore	SpringerLink	Wiley	Total white	Google Images	Google Patents	Grand total
Initial search	183	82	266	31	562	100	100	762
Without duplicates	181	79	263	31	554	100	100	754
First phase	9	9	26	3	47	21	12	80
Second phase	7	8	13	3	31	11	7	49

Table 4.1: The amount of sources from each database after each step in the literature review process.

4.2 Pattern Extraction

When the final set of component models had been compiled the extraction of patterns from them could begin. At this point, a good understanding of the component models had been attained thanks to the filled-in summary in the data extraction form, which sped up the pattern extraction process significantly. The pattern extraction process was done in an iterative manner where each component model was looked at many times in total to minimise the risk of missing patterns. Each component model was printed on a piece of paper to make cooperation easier and so that notes could more easily be made, and then sorted into three piles for better organization, one for each type of system (whole system, part of system, and pipeline).

Iteratively, each component model would be looked at by both authors to find similarities between them. These similarities, or aspects of the component models that *could* be similar between component models, were noted down and circled on the printed-out component models. A few examples of how this was done can be seen in Figure 4.3, which shows how circles, arrows, and other notes could effectively be used to aid in this matter. All similarities were also inputted into a document to maintain organisation. However, it is worth emphasising that at this point, it was not patterns that were extracted, only high-level similarities between component models were documented. The plan was to then later break down these similarities into patterns with distinct definitions. A few examples of these similarities that were noted down at this point were *circular improvement* (when there was a loop in, e.g., training or runtime of the model), *orchestrator* (a common component name across multiple component models), and *running multiple models* (when the component model contained multiple ML-models). Each component model was looked at at least once after every new found similarity.

After no more similarities could be found, the previous literature reviews on ML or AI architectural design patterns were examined, i.e. the ones by Washizaki et al. [39] and Heiland et al. [40]. Up to this point, while these sources had been read, the patterns themselves had not been looked at to reduce bias. This was done to see if their identified patterns could be found in the component models, which, indeed, some could. Furthermore, some of the identified similarities that had been noted were found to be established patterns with an already given name, problem, and solution. This resulted in some extracted patterns being already established patterns, while some were, as far as we know, new and coined by the authors.

At this point, the found similarities were looked at one at a time to distil them into patterns with distinct definitions and problems it solves. If some similarity did not solve a distinct problem it was removed. For instance, when all component models that had *circular improvement* were looked at simultaneously, it became clear that there were three kinds of circular improvement present: Training, system, and runtime evaluation and improvement. Training evaluation and improvement was not considered a pattern since it is a fundamental aspect of ML-enabled systems and does therefore not solve a distinct problem. The other two were considered as patterns since they solved a distinct problem. For the already established patterns,

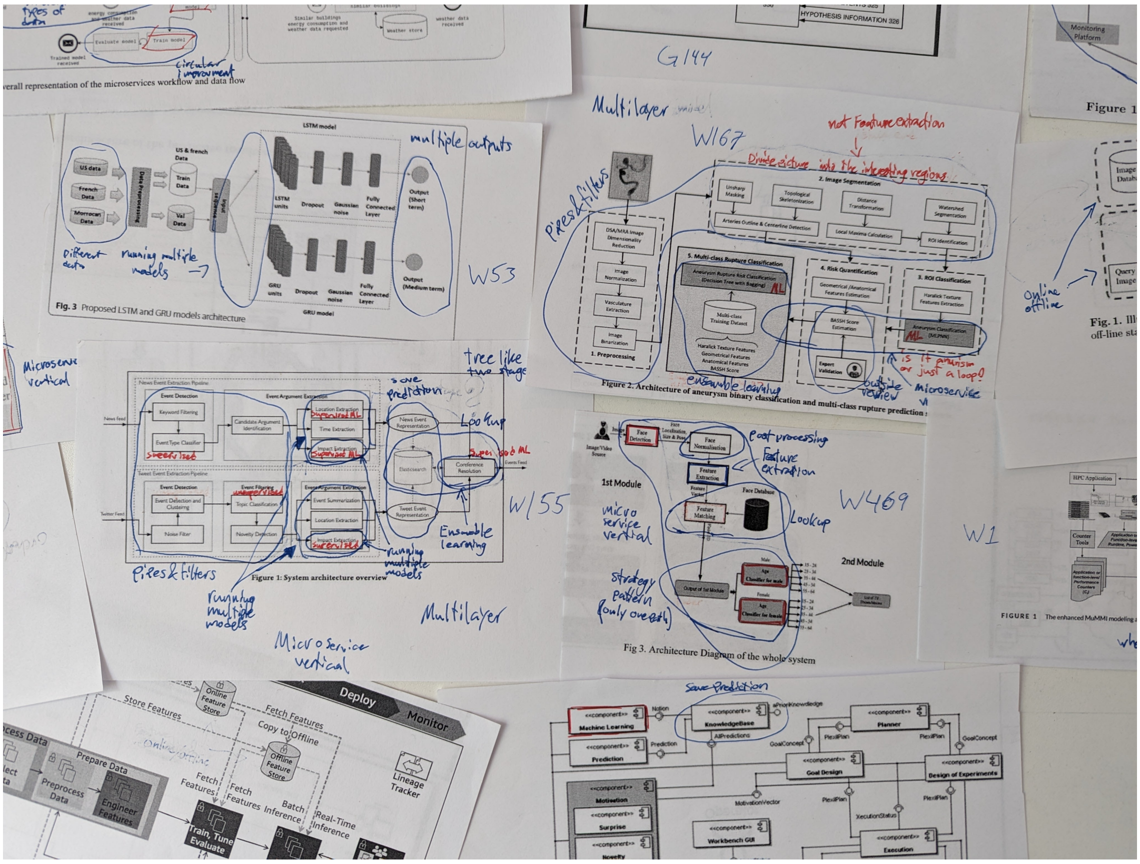

Figure 4.3: Some examples of the notes that were made on the component models.

the problems a pattern solves can be directly found through the related literature, e.g., Heiland et al.'s MLR [40]. Sometimes, the pattern can be connected to quality attribute impact already from the given problems it solves. For the new patterns, though, the problems a pattern solves have to be constructed by the authors. This was done by delving into the design decisions of the component models that contained the pattern by reading the sources. In the end, a total of 14 patterns were found in the component models. Which patterns were found in which component models can be seen in appendix B.

Recall that this study only considers patterns that appear in at least two component models, as described in section 1.3. This pattern extraction strategy will not be exhaustive, since there might be patterns that only occur once, patterns that were simply not found, and even patterns that did not occur in the set of component models found in the MLR. But as mentioned, the goal of this thesis is to find patterns used in ML-enabled systems and evaluate them based on quality attributes, and thus finding a set of patterns that is not exhaustive is acceptable since the goal is the evaluation of used patterns, not the list of patterns itself.

4.3 Interviews

As section 3.3 highlighted, evaluating the impact design patterns have on quality attributes is difficult, but possible. The selected evaluation methodology was expert opinion, one of if not the best for this task [32].

More specifically, semi-structured interviews were planned and conducted to elicit data that could be used to answer RQ2 as well as RQ3. This method was selected partially because when asking experts on pattern and quality attribute impact, they can simultaneously give input on the pattern connection, thereby giving data to answer both RQ2 and RQ3.

Semi-structured interviews give the right balance between freedom and structure when it comes to the questions asked, and let the interviewees think freely about what QAs stick out the most to them, as well as how large of an impact and the reasons why. Structured interviews could be problematic, at least if specifically asking about all QAs for a given pattern since the order or the QAs as well as their (somewhat large) count could introduce biases.

During the planning and conducting of the interviews, Hove and Anda's recommendations for semi-structured interviews in empirical software engineering research [47] were taken into consideration. These recommendations were also followed regarding what to report from the interviews.

4.3.1 Planning the Interviews

Interviewees were recruited via email. The staff of the Data Science and AI department (DSAI), as well as the staff of Interaction Design and Software Engineering (IDSE) within the area of AI Engineering, were asked to participate, both at the joint

Computer Science and Engineering department at Chalmers University of Technology and Gothenburg University. These were selected because they are, as far as we know, the only experts on AI or ML in academia within a close enough proximity for us to have a physical interview with. The emails contained an overview of the research topic to explain the purpose of the interview and were formulated in an attempt to interest as many applicable interviewee candidates as possible.

An attempt to reach out to practitioners of ML-enabled systems within industry was also made, namely to AI Sweden[2]. However, due to a lack of replies from them and time constraints on us, this idea unfortunately had to be abandoned.

Out of the 65 asked from DSAI and 15 from IDSE, 7 and 4 respectively agreed. Unfortunately, one person from IDSE fell sick without the possibility of rescheduling within our time frame, giving a total of ten interviews.

4.3.1.1 Interview Script

To help in conducting the interviews, an interview script was made which was followed during the interviews. The full script can be seen in appendix D, but a summary of it is provided below.

As [47] suggested, the interviews were opened with an explanation of the interview's purpose and assurance that what they say is anonymous. Interviewees were encouraged to speak their minds since there were no right and wrong answers.

1. Provide some general information, including an introduction to the thesis and why their input is needed for the study.

   If the interviewee had no questions after the general information, this is where we would ask for permission to start the audio recording.

2. Ask them to briefly share their experience with ML and ML-enabled systems.

3. Show the quality model (Figure 4.5) and ask if they have any questions regarding it.

4. Show a pattern.

   (a) Explain the definition of the pattern and an example of it in use.

   (b) Ask them if they have seen or used something like this before.

   (c) Ask which quality attributes they think the pattern affects and why.

   For each quality attribute connection, let them rate on a scale, seen in Figure 4.4, how big of a positive or negative impact the pattern will have on the quality attribute.

5. If time is available, go back to step 4. Otherwise, end the interview by thanking them for their participation.

[2]`https://www.ai.se/sv`

Figure 4.4: The scale that was provided during the interviews for the interviewee to rate the QA impact on a given pattern, ranging from a strong negative impact to a strong positive one.

4.3.1.2 Pilot Interview

To test our guidelines and practice our interviewing technique we held a pilot interview with a student from the Data Science and AI master program with a background in software engineering. During this interview, we realised that if not prompted otherwise, the interviewee may go too in-depth on the quality attributes and try to find things to say about all of them, which is not the goal of the interviews. While you can try to find something to say about every quality attribute, since the goal is to find the bigger impacts — since these impacts will be the most useful as results — to make time more effective, we, therefore, learned that we needed to explicitly say to the interviewees that they do not have to go through all QAs.

4.3.2 Conducting the Interviews

A few days before each interview, an email was sent to the interviewee containing the quality model, with a request for them to prepare for the interview by looking at it so that they would be prepared on which quality attributes appear in it. The image that was sent can be seen in Figure 4.5.

Before each interview, a set of patterns were selected for the interview. For the first interviews, simpler patterns were chosen to allow for more focus on conducting the interviews and to build confidence in interviewing. For later interviews, the patterns that had been talked about the least so far in the interviews were chosen. There was one exception to this, which was the system architecture patterns and system evaluation and improvement. Since these patterns are more within the field of software architecture than machine learning, it was chosen to not show these patterns

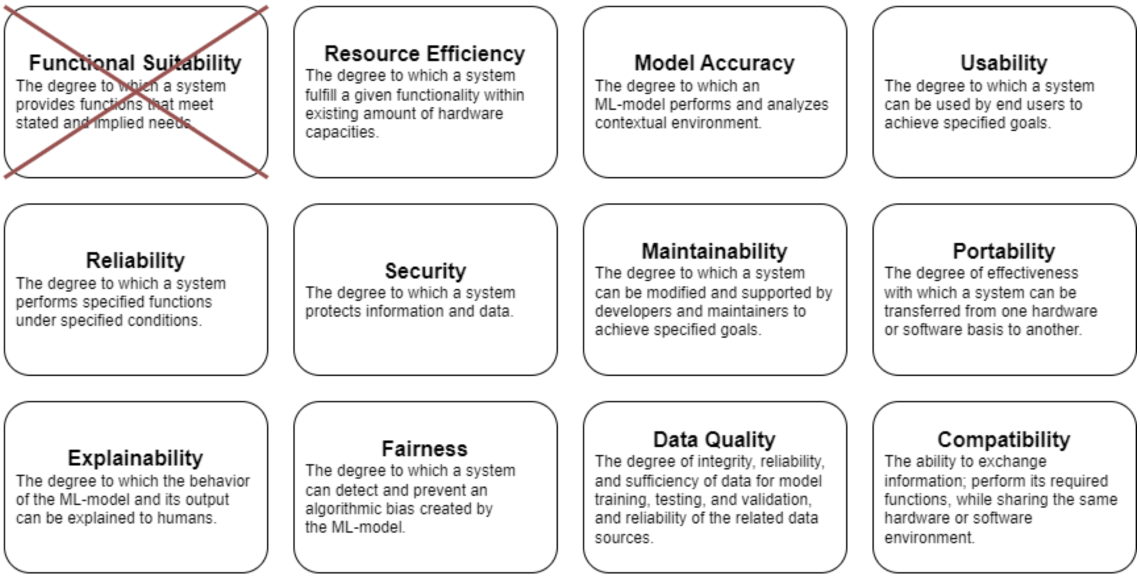

Figure 4.5: The format of our quality model that was used for the interviews.

to the interviewees from DSAI since they are not within that field of expertise. Table 4.2 shows the experts, labelled from E1 to E10, which department they were from, as well as which patterns were discussed in their interview.

Interviews were held in the nicest rooms bookable by the authors near the interviewee's office, except three interviews. For two of them, the interviewee offered to meet at their office, and one had to be conducted online. While meeting physically was preferred, not least because it enables more direct conversation, an online interview was better than no interview.

The Google Recorder app was used to get auto transcription, which saved a lot of time. However, each transcription was later manually cleaned and verified since there were always some miss-transcribed words that occurred in the auto transcription. This approach could not be used effectively for the interview conducted online. Instead, this one was transcribed by hand.

The interviews were planned to take roughly half an hour. This amount of time let us get through two to five patterns, with an average of 3.4 patterns per interview. An attempt was made to evenly go through the 14 patterns.

Both authors were present during the interviews. One had the role of script reader — the one who was holding the interview. The other author took notes and handled the audio recording, but still asked follow-up questions if relevant. This is with the exception of one interview, where one of the authors was unavailable.

4.3.3 Interview Data Extraction

To extract the data from the interviews we went through the transcriptions of the audio recordings to summarise the main talking points and their associated impact ratings for each pattern and quality attribute combination. This is an interpretive

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
Ensemble learning		●			●		●			
Parallel independent models	●		●				●			
Sequentially dependant models		●		●					●	
Situation-based model selection	●				●		●			
Data transformation				●		●				
Feature extraction			●			●				●
Post processing			●							●
Prediction cache				●		●				
Multi-layer pattern							●		●	
Orchestrator							●	●		
Server-side ML model								●	●	
Expert validation	●				●					●
Runtime eval. and improvement			●							●
System eval. and improvement								●	●	
DSAI	●	●	●	●	●	●				●
IDSE							●	●	●	

Table 4.2: The ten experts, labelled from E1 to E10, and the patterns that were discussed with them, together with their division.

content analysis methodology [48] where the focus does not solely lie on what the interviewees said, but also what they implied. A more rigorous methodology, such as a thematic analysis [49], could have been better suited if the amount of data collected for each pattern was larger.

This was then cross-referenced with the notes taken during the interview so that they matched and no impacts were missed. During this step, the data was also filtered to remove implementation-specific impacts, second-order impacts, and to move misplaced impacts due to misinterpretation of the quality attributes.

The motivation is the most important aspect of the interview data since the impact on the quality attribute is based on it. Therefore, ratings without a motivation were removed from the data because the reader must be able to see how the pattern impacts the quality attribute. Motivations without a rating were not removed, however, since the readers can assess the impact on the quality attribute themselves, and removal would obscure important information from the readers.

The motivations were then compared and, if deemed to be the same, combined. The individual ratings and the experts who highlighted the impact were documented and will be presented along the motivation itself.

5

Results

The results of the multivocal literature review — the 49 component models — can be found in appendix A or seen in a publicly made repository[1]. The rest of this section will detail the results of the study in the following way: First, the patterns that were extracted from the component models. Second, the association to, and impact of, the patterns on the quality attributes, as said by experts in interviews.

5.1 The Patterns

From the component models produced by the MLR, a total of 14 patterns were found. These can be seen in Table 5.1 together with how often each pattern occurred in the 49 component models. To better comprehend and present the patterns, they have been grouped into four overarching categories depending on common aspects between the patterns: Multiple model patterns, Data handling patterns, System architecture patterns, and Validation patterns.

Occurrences	**Pattern**
8	Data transformation
5	Ensemble learning
3	Expert validation
5	Feature extraction
5	Multi-layer pattern
5	Orchestrator
7	Parallel independent models
7	Post processing
2	Prediction cache
4	Runtime evaluation and improvement
6	Sequentially dependant models
5	Server-side ML model
2	Situation-based model selection
3	System evaluation and improvement

Table 5.1: The found patterns and the number of times they appeared in the extracted component models.

[1]https://github.com/The-Impact-of-Design-Patterns-on-Quality-Attributes-in-ML-Enabled-Systems/Component-Models

The rest of this section will detail the patterns. Each pattern is defined clearly and concisely, to make the pattern easy to understand and use. Each pattern has also been analysed to find the problem(s) that it aims to fix, an example from the extracted component models and, if relevant, background information of the pattern.

5.1.1 Multiple Model Patterns

These patterns concern how a system can utilise multiple models in different ways to achieve different goals. What is common between all patterns in this category is that, as the name suggests, the system contains at least two ML models.

5.1.1.1 Ensemble Learning

Definition: A system running multiple models, combining them into one prediction. This can either be done by multiple models running on the same data, different parts of the same data, or in sequence.

Problem it solves: Having multiple models can allow the models to look at the same problem from different angles, allowing the system to get a more holistic view before coming to a conclusion.

Background: Ensemble learning is an established technique for combining the output of multiple models to increase model accuracy and resilience [50]. There are three different, main approaches on how to do ensemble learning [51]:

- Bagging — fitting many decision trees on different samples of the same dataset and averaging the predictions.
- Stacking — fitting many different model types on the same data and using another model to learn how to best combine the predictions.
- Boosting — adding ensemble members sequentially that correct the predictions made by prior models and output a weighted average of the predictions.

We chose to keep these as one pattern since they all solve the same problem, just in a slightly different manner.

Example: Figure 5.1 depicts a system trying to find *Botnet* domain names [52]. Botnet domains have names either of random characters or random words. The system looks for these features in parallel and then combines the answers from the two models to give a prediction if it is a botnet domain.

5.1.1.2 Parallel Independent Models

Definition: Running multiple independent models on the same input data to find different characteristics using different specialised models to get different outputs.

Problem it solves: When doing complex analysis on the input data it can be hard to get the model to differentiate between different types of information within the data. Therefore, running multiple models specialised in finding their niche characteristics within the data can allow the system to more reliably find these characteristics.

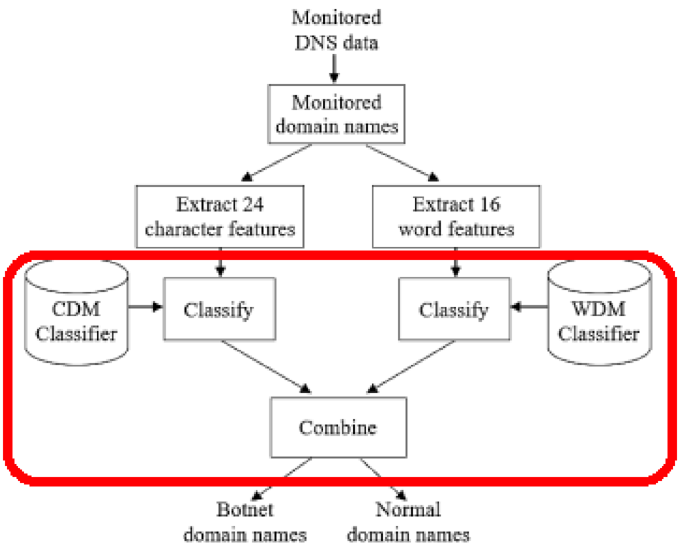

Figure 5.1: Example of the Ensemble pattern taken from [52] with the pattern circled in red.

Background: This pattern is known as *Microservice horizontal pattern* in the work by Heiland et al. [40]. The name was changed in this work to reduce confusion and widen its scope. Differentiating between "horizontal" and "vertical" can be confusing (e.g., the authors of this thesis associate horizontal with sequential and vertical with parallel), which is one of two issues with the original name. The second issue is that to achieve the benefits of using this pattern you do not have to use microservices, which is why that word was removed in this work. Usage of this pattern means that the models are independent as well, which is yet another change from [40] which states you can "acquire multiple predictions, or an integrated prediction". In our work, we separate parallel independent predictions from parallel combined predictions (where the integrated prediction depends on multiple models), which is instead classified as ensemble learning.

Example: When doing image processing, finding and differentiating between different types of objects can be tricky for an ML model. Therefore, it could be easier and more reliable to have multiple models searching for different things. An example of this can be seen in the top left of Figure 5.2, where the image processing searches for traffic lights, lanes, and obstacles independently on the same input image.

5.1.1.3 Sequentially Dependant Models

Definition: Running multiple ML models in sequence, each with the distinct responsibility of solving a smaller step in the process of solving a greater problem.

Problem it solves: Some problems are complex enough that a single model could have a hard time finding the right conclusion to the problem. One way to solve that is to break the problem down into smaller, more manageable steps that build upon each other. In ML-enabled systems utilising this technique, multiple models are employed sequentially that build upon the output from the previous one. The

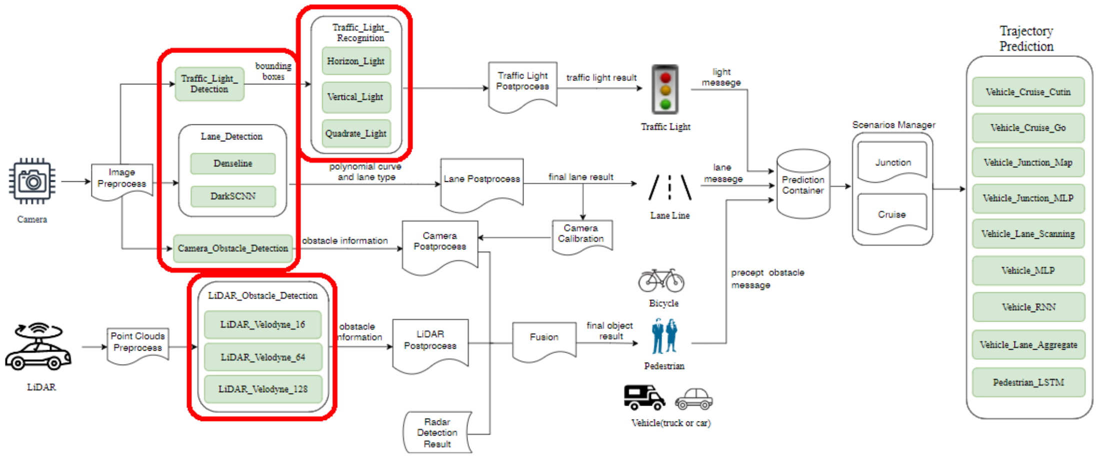

Figure 5.2: Example of the Parallel independent models pattern taken from [53] with three separate instances of the pattern circled in red.

final model in this chain then makes the final prediction, based on the work of the feeder models, which is the sought answer to the greater problem.

Background: This pattern is known as *Microservice vertical pattern* in the work by Heiland et al. [40]. For the same reasons as Parallel independent models in section 5.1.1.2, this pattern was renamed and clarified.

Example: An example of this can be seen in Figure 5.3, where they extract a face from an image and then, after some data processing to find the gender, input the face into an age detection ML model to retrieve the age of the person.

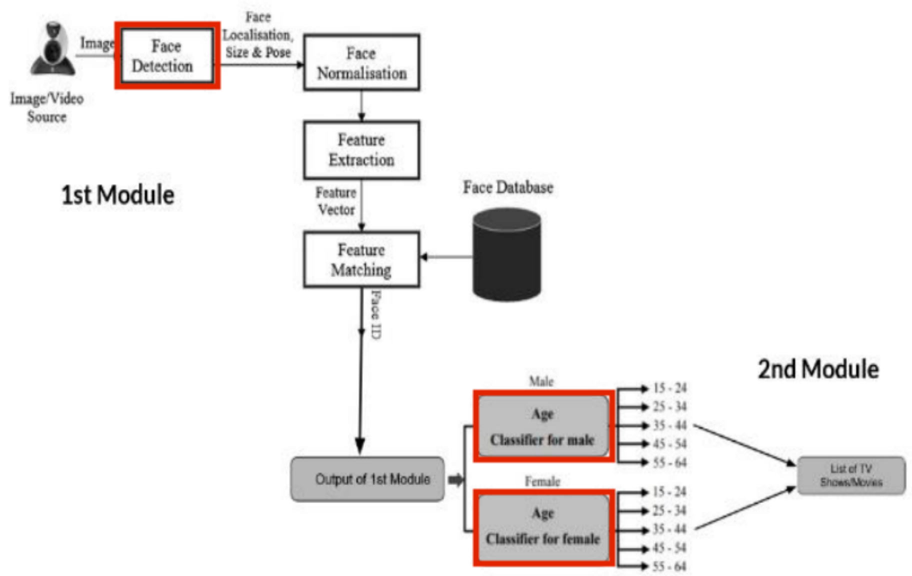

Figure 5.3: Example of the Sequentially dependant models pattern taken from [54], with the machine learning models circled in red.

5.1.1.4 Situation-Based Model Selection

Definition: When the system has multiple models to choose from and picks one of them to use based on the situation.

Problem it solves: The models can be tailored to a specific situation or state so that they can perform their task better since there are known variables that do not need to be incorporated in the ML model.

Background: This pattern can be seen as a special case of the strategy pattern [55] since the models are interchangeable and can be changed at runtime. The big difference between them is that this pattern is not a general pattern, as the strategy pattern is, but rather an ML-specific pattern with a more specialised goal.

Example: As seen in Figure 5.4, the "Scenarios Manager" has nine different models for different scenarios or states that a car can be in, with tailored models for each state to predict the optimal trajectory.

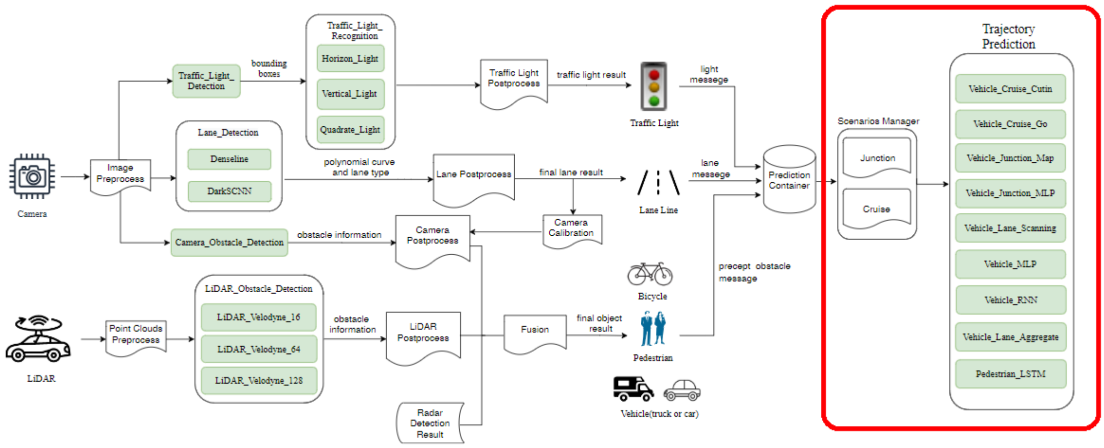

Figure 5.4: Example of the Situation-based model selection pattern taken from [53], with the pattern circled in red.

5.1.2 Data Handling Patterns

These patterns concern how input and output data is handled. This could be how data is used or handled before going into the ML model or concern what happens to the data the model outputs.

5.1.2.1 Data Transformation

Definition: Data is transformed through a series of procedures, which produce incremental results to transform the data into the sought type of input for the model.

Problem it solves: The raw data might not be in the correct format for the ML model and needs to be broken down or changed so it fits the model. Heiland et al. also describe how the pattern can improve the flexibility of the system [40].

Background: This pattern is known as *pipes and filters* in the work by Heiland et al. [40]. But, as we believe this name does not convey the goal of the pattern very well, we have decided to change it and believe the name *data transformation* describes it better.

Example: As seen in Figure 5.5, the input, in this case pictures of blood vessels, are taken and filtered in different steps. The transformation is first to find the blood vessels, and then extract the parts of the blood vessel that might be an aneurysm, which is done via image processing and not ML.

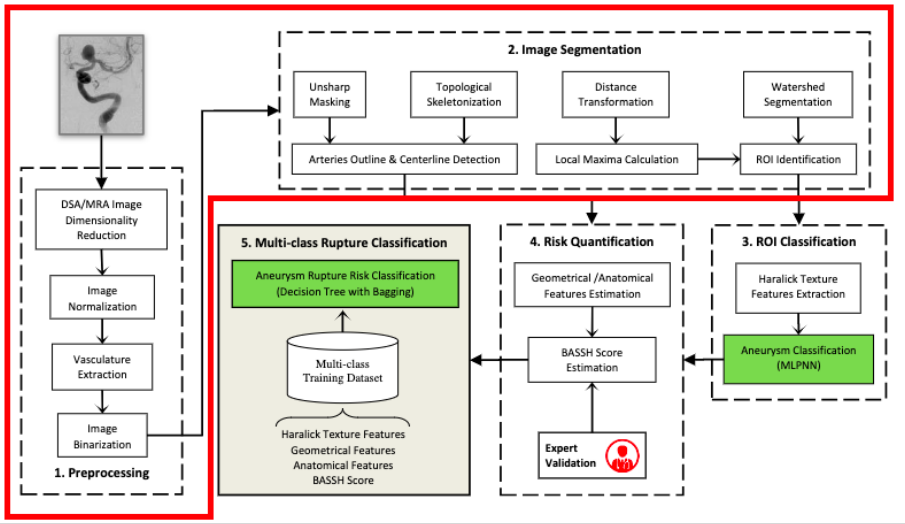

Figure 5.5: Example of the Data transformation pattern taken from [56] with the pattern circled in red.

5.1.2.2 Feature Extraction

Definition: When the input for the model first goes through a component that extracts a set of features within the data instead of using the raw data as input for the model.

Problem it solves: A model running on a dataset with large entries can become very complex, and running it on features in the data allows it to be of a more manageable size. This also allows pieces of the data to be hidden from the model, if they are deemed unimportant or of negative impact.

Background: Feature extraction is a well-established technique within machine learning [57] for decreasing the size of the feature space without losing information about the original feature space [58].

Example: In Figure 5.6 we can see a system that takes in domain names and extracts two sets of different features that it runs through the models. In this case, it is two different models and their output is combined, but it could be any type of ML model and all features can be input for the same model.

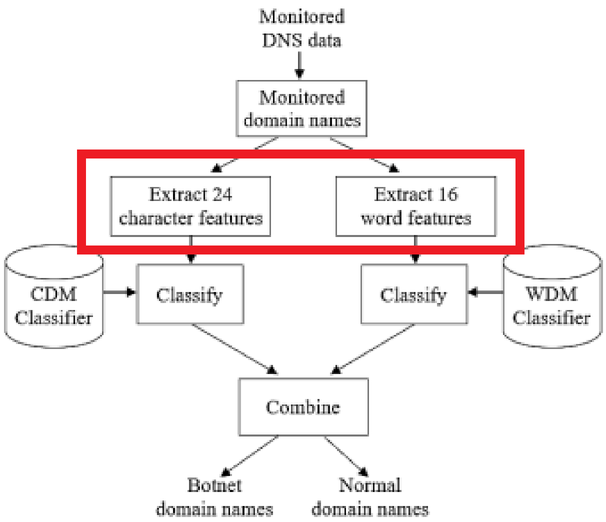

Figure 5.6: Example of the Feature extraction pattern taken from [52] with the pattern circled in red.

5.1.2.3 Post Processing

Definition: If the output of the model is not in the format wanted by the system, it is fed into a component that processes the model output to the desired format for the system.

Problem it solves: The output from the model is in the wrong format, or some information or features need to be reconstructed since they can be lost in the model.

Example: Figure 5.7 shows an example of post processing from a Bose Inc. patent. After the ML component has filtered out background noise from a microphone input, a component is used to reconstruct the sound of the person speaking into the microphone. Thanks to the reconstruction component the background noise is removed but the sound of the speaker can be preserved.

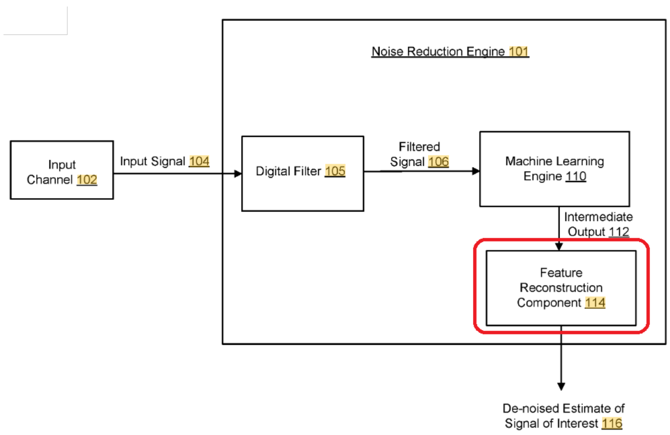

Figure 5.7: Example of the Post processing pattern taken from [59] with the pattern circled in red.

5.1.2.4 Prediction Cache

Definition: The input and/or prediction is saved so that it can later be queried to see if it is unique.

Problem it solves: If the input to the model can be repeated multiple times the corresponding output can be stored, meaning the model does not have to make duplicate predictions. This strategy can also be used in real-time data fetching systems to remove duplicate predictions which can help prevent unnecessary output overload.

Background: This pattern is taken from Heiland et al.'s work [40]. In their work, a similar pattern called *Data cache* is also presented. While prediction cache in Heilands et al.'s work is specific to predictions and data cache is more towards preprocessing of data, here they are merged since there is no significant architectural difference between the two.

Example: In Figure 5.8 we can see an example of the prediction cache pattern. This system takes real-time news and Twitter info and boils that info down to a simpler version. Since multiple sources can talk about the same event there is a component at the end that saves each prediction, in this case, an event representation. These are then checked if any of the ones made before are close enough that they represent the same event to make sure the event feed is not flooded each time an event occurs.

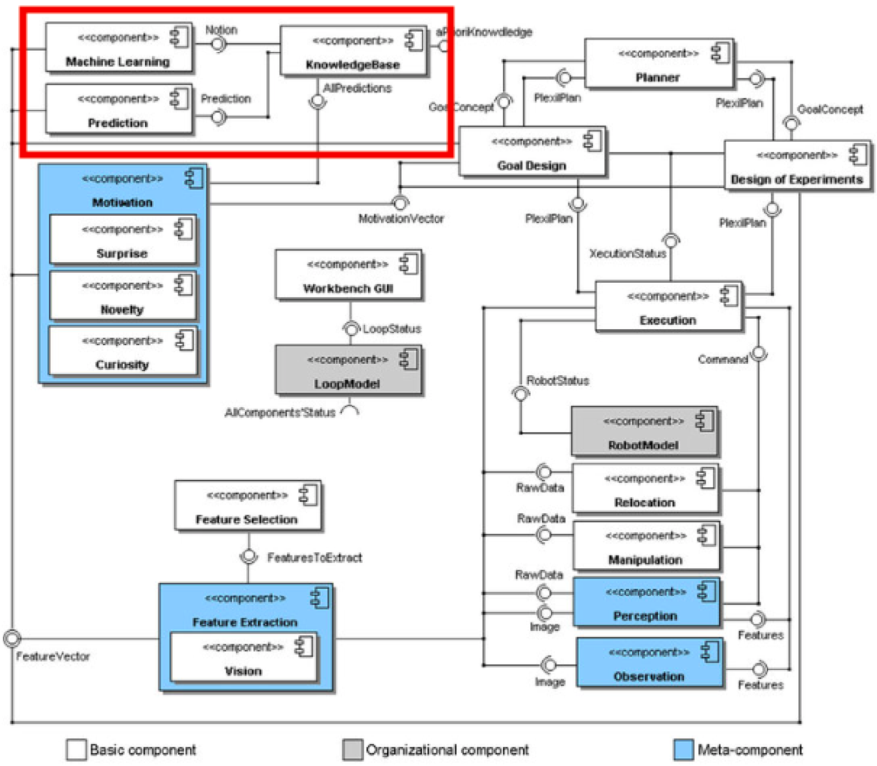

Figure 5.8: Example of the Prediction cache pattern taken from [60] with the pattern circled in red.

5.1.3 System Architecture Patterns

These patterns concern how components are structured on a high level and how they communicate with each other. These patterns do not concern the specifics of the models themselves. It should however be highlighted that since these patterns were found in ML-enabled systems, they are suitable for this type of system even if they do not concern the model-specifics.

5.1.3.1 Multi-Layer Pattern

Definition: The system is divided into different layers which have clearly defined purposes where each layer communicates only with its closest neighbours.

Problem it solves: To separate the system into layers helps in creating *separation of concern* as well as *low coupling, high cohesion* which are two well-known software design principles [61].

Background: This pattern is taken from Heiland et al.'s work [40], where it is described as a traditional pattern (i.e. a pattern that appears outside AI/ML contexts) that was mentioned in a large number of sources describing AI or ML design patterns. This pattern is also known as *Separation of Concerns* or *Multi-Tiered Architecture* [40]. The definition in this work has been refined and broadened compared to the problem and solution given by Heiland et al., where each layer should have different levels of abstraction (which here is changed to purposes) and only communicates to its closest neighbour (which here is changed to neighbours, in plural).

Example: In Figure 5.9, we see an example of a system employing the multi-layer pattern. This system has two separate paths, one for Twitter and one for news feeds where it looks for new events. The information from these is passed through a series of layers that first detect events, secondly filter the events (only for Twitter), then find the information about the event that the system wants, and finally look for uniqueness before publishing to the event feed.

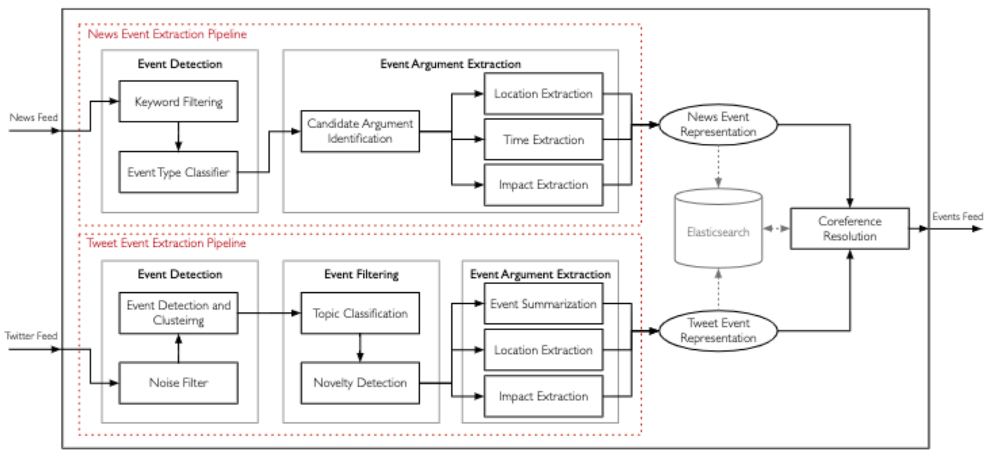

Figure 5.9: Example of the Multi-layer pattern, taken from [62].

5.1.3.2 Orchestrator

Definition: A central component that handles and initiates communication between the machine learning-related components, including but not limited to, data handling, training, evaluation, and prediction requests.

Problem it solves: Since ML-enabled systems can contain quite a lot of different components and have to handle large amounts of data, it can be beneficial to have a central component that handles the coordination between all the components so that they can be dedicated to solving their task.

Background: This pattern is similar to the *mediator pattern* in the work by Heiland et al. [40]. But in contrast to the mediator pattern, the orchestrator is the initiator of the contact with the components while in the mediator pattern the components communicate with each other through a mediator. The problem they solve is therefore different, meaning they should be separate patterns.

Example: As seen in Figure 5.10, the orchestrator connects all the components. It will manage the different models that the model builder will create with the data supplied by the orchestrator, as well as evaluation, supplying models to the server, etc.

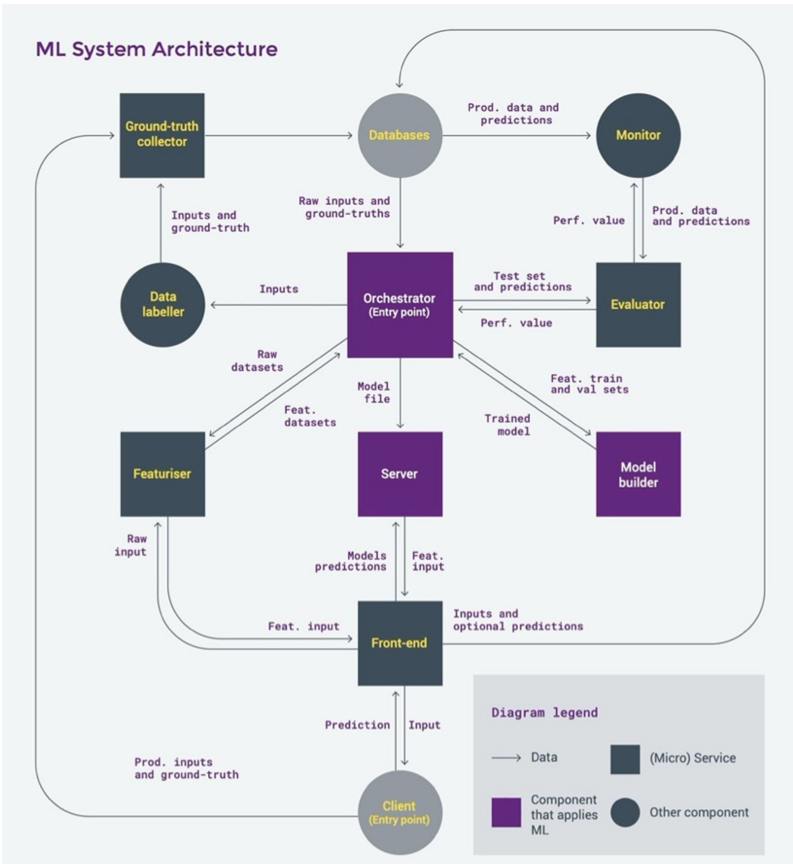

Figure 5.10: Example of the Orchestrator pattern taken from [63], where the orchestrator itself can be seen roughly in the middle.

5.1.3.3 Server-Side ML Model

Definition: One component has the role of server and allows components or sub-systems to take the role of client, initiating a connection with the server to obtain a prediction from its model.

Problem it solves: If an ML component is used from multiple components or sub-systems, this pattern allows the system to have the model in one place instead of scattered around the system.

Background: This pattern is known as *Client-Server* pattern in Heiland et al.'s work [40]. The new name, Server-side ML model, more clearly describes what the pattern represents, emphasising that it is in fact the ML model that is on a server, centralising it for an arbitrary number of clients to use.

Example: In Figure 5.11, an example of the pattern can be seen where the sub-system at the bottom of the figure acts as a server and provides its machine learning service to the client at the top.

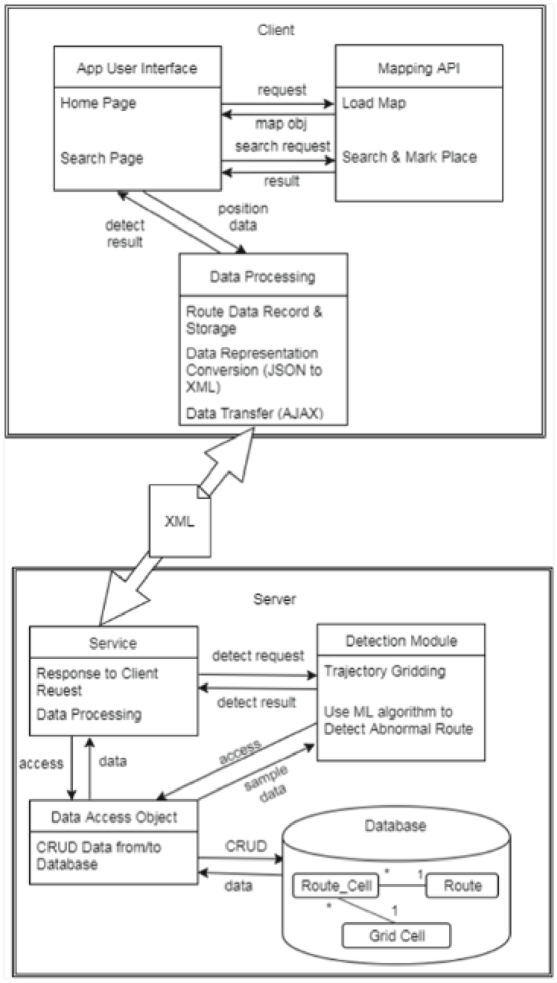

Figure 5.11: Example of the Server-side ML model pattern taken from [64].

5.1.4 Validation Patterns

These patterns concern validation within the systems in some way, either by evaluating an ML model directly or an application within a larger system. By having some form of validation in these ways, an improvement to key aspects of the system is intended.

5.1.4.1 Expert Validation

Definition: A system where a human domain expert validates a critical step to ensure it fulfils certain criteria.

Problem it solves: Since the output of an ML model is a prediction and not the truth, it can never be trusted as such. This means some systems with low margins for error, such as safety-critical systems, can have issues with making sure the model makes the right decision every time. To combat this, some systems have an expert validate a critical step of the process to make sure it works as intended.

Example: An example of the expert validation pattern can be seen in Figure 5.12, a simple component model depicting ensemble learning with expert validation. Here, the expert validates the consensus checking between the models.

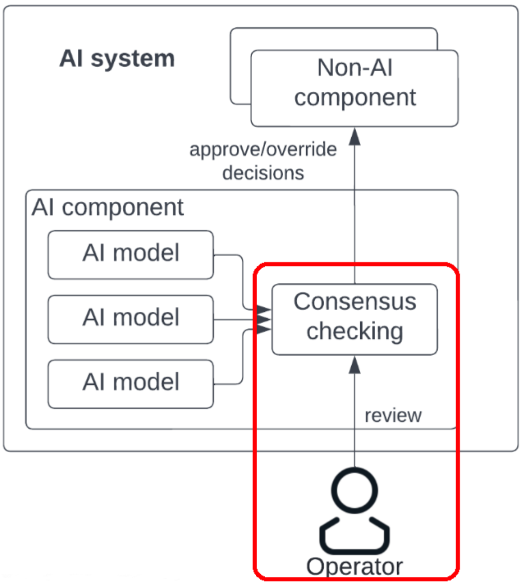

Figure 5.12: Example of the Expert validation pattern taken from [65] with the pattern circled in red.

5.1.4.2 Runtime Evaluation and Improvement

Definition: A system that monitors and evaluates the ML model during deployment, and makes improvements to it either by retraining, tweaking the model, or creating a new model with some changes.

Problem it solves: As a machine learning model is only as good as its training data, a model will perform at its best just after training. The longer after the training time goes, the more the environment might change compared to the training data the model trained on and make the predictions less accurate. This problem is known as model drift and can be solved by updating the model during deployment to make sure it follows the changes of what it is supposed to predict.

Example: In Figure 5.13 we see an example of the runtime evaluation and improvement pattern from the Amazon Web Services documentation. As seen, a model is trained and deployed, and then it is monitored and adjusted, either by tuning it or by going back to adjust the training data.

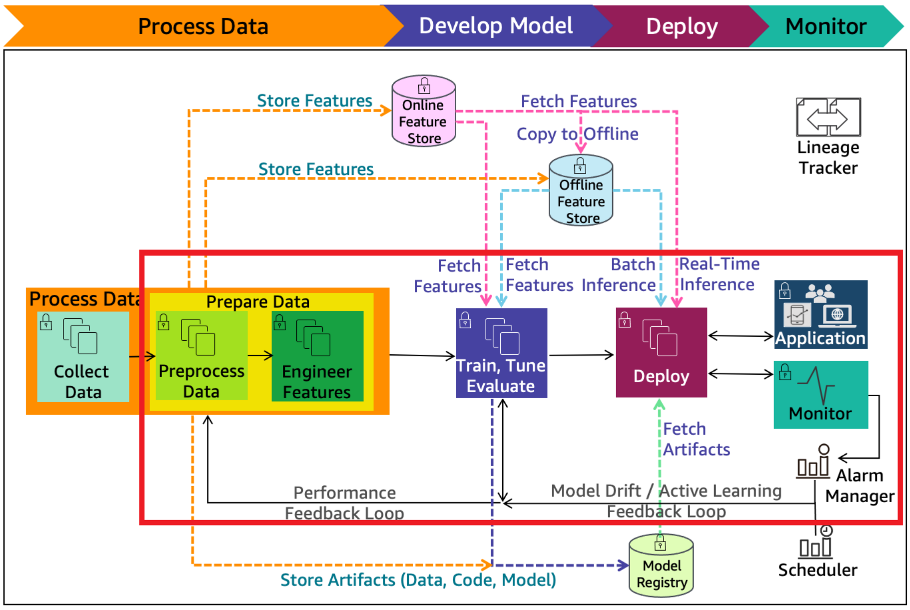

Figure 5.13: Example of the Runtime evaluation and improvement pattern taken from [66] with the pattern circled in red.

5.1.4.3 System Evaluation and Improvement

Definition: A system that is monitored by an ML model to evaluate the system itself and to make iterative changes to the system based on the feedback from the model.

Problem it solves: Every system is surrounded by an environment. If this environment changes, it could bring problems to the system. To combat this, the system can be continuously evaluated by an ML model that can catch changes in the environment and/or faults in the system instead of them impacting the system negatively.

Example: In Figure 5.14, an example of the system evaluation and improvement pattern can be seen, where the evaluation of the larger system happens at the top right. The middle layer decides the best adaptation to try and then implements those changes before repeating the process.

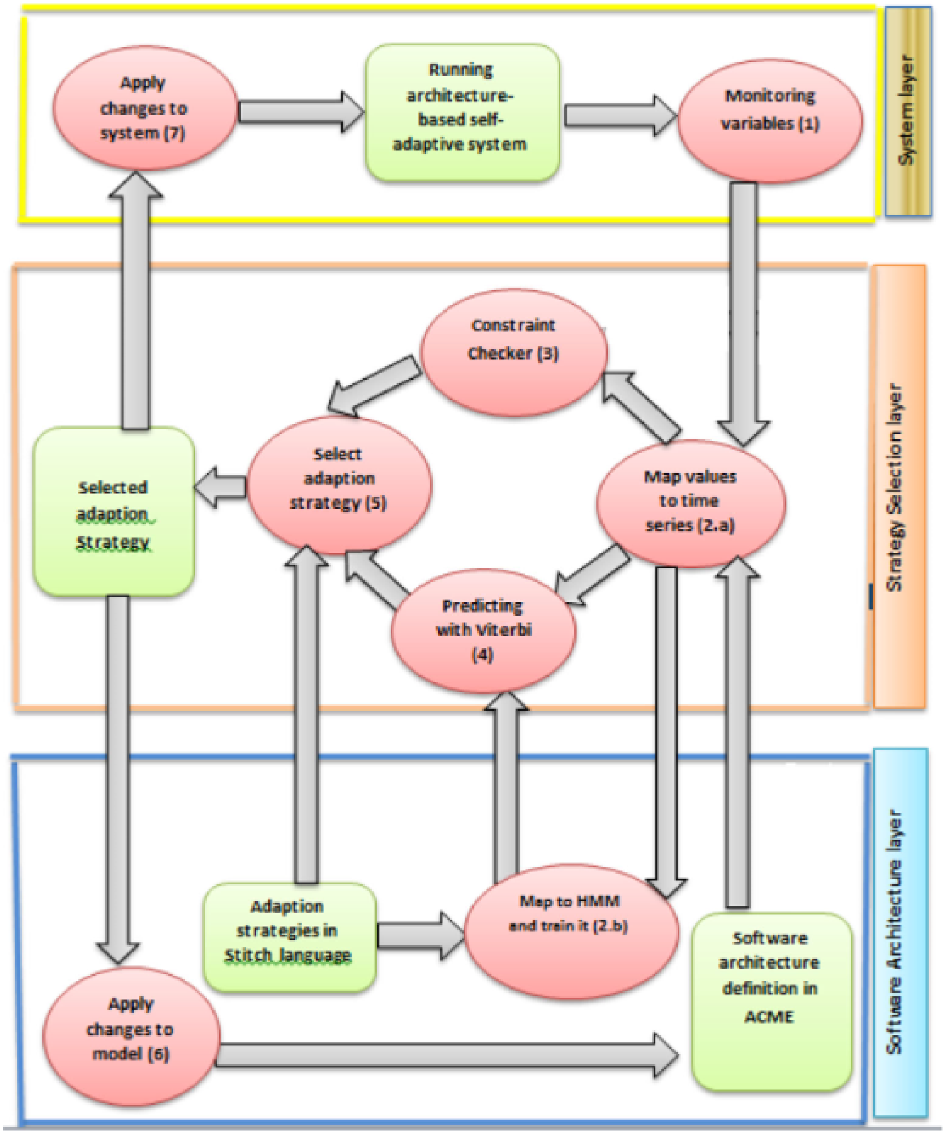

Figure 5.14: Example of the System evaluation and improvement pattern, taken from [67].

5.2 Quality Attribute Association and Impact

In this section, the results of the interviews are presented. Table 5.2 provides an overview of the results, but more nuanced motivations are found in the following sections. The table shows the association between patterns and quality attributes as said by the interviewees, thereby answering RQ2. However, RQ3 is not answered in the table since the extent of an impact for each association has to be read in the sections below which detail the individual impacts and corresponding motivations.

Pattern	~~Func. Suitability~~	Resource Efficiency	Usability	Reliability	Security	Maintainability	Portability	Explainability	Model Accuracy	Fairness	Data Quality	Compatibility
Ensemble learning		+*		+		−	−		+	+		
Parallel independent models		−		+		+*	−		+			
Sequentially dependant models		+		−		−		−*	−	−		+
Situation-based model selection		+*	+			+			+	+		
Data transformation		±			+	−	−	−	+	−		
Feature extraction		−		+*	+			+*	+	+*		
Post processing			±	+						+		+
Prediction cache		+		−	−		−					
Multi-layer pattern						+	+					
Orchestrator		+*		−	+	+	+					+
Server-side ML model		−			+*	+	+					+
Expert validation			+	+		+				±		
Runtime eval. and improvement				+		−					−	
System eval. and improvement		+*		+	+	−						−

Positive impact	+	Balanced positive and negative impact	±
Negative impact	−	Impact is not unequivocal	*

Table 5.2: Quality attributes associated with the patterns, answering RQ2, as highlighted by the experts in the interviews.

Below are the motivations given by the interviewees for every impact. The impact of each motivation will be indicated on the left side, the experts that gave that motivation in the middle and the motivation itself on the right. The experts will be sorted by their identifying number from Table 4.2, and their corresponding rating displayed in the same order, as shown in the example below, where expert *a* gives rating 2, expert *b* gives the span “1 to 2”, and expert *c* gives the motivation without a rating.

2, 1–2, ? E*a*, E*b*, E*c*: Here is a motivation why the pattern impact this quality attribute.

An important aspect to note is that since all systems are unique the applicability of these results will vary depending on system, since not all patterns are suitable for each system. This is especially important considering that the system's requirements will govern which patterns are used to a larger extent than the quality attributes that are important to the system. These results are rather the quality attribute impact one can in general expect if the pattern is implemented correctly in a suitable system.

5.2.1 Ensemble Learning

Resource efficiency:

2, 2	E2, E5:	Each model can be trained on a subset of the data. This will speed up training, increasing the performance of the system.
−1	E2:	If the task is specialised, one specialised model will do predicting more efficiently.

Reliability:

2, 2–3, 2	E2, E5, E7:	Prediction is more trustworthy when done by multiple models, since if one makes a bad prediction the others can make sure the end result is correct.

Maintainability:

−1,−1,−1	E2, E5, E7:	As more models are introduced into the system there will be more maintenance needed.

Portability:

−1	E7:	When using ensemble learning we are bound to using models that support ensemble learning. Therefore, if the system is to be ported there needs to be ensemble learning supported models that support the new hardware or software.

Model accuracy:

2, 2, 2	E2, E5, E7:	If the models are different but trained on the same data we can get different nuances of the same info, and thus get a more nuanced prediction.

Fairness:

1	E7:	Different models provide different views on the same problem, this will reduce the way biases propagate throughout the system.

5.2.2 Parallel Independent Models

Resource efficiency:

−2	E7:	Since we introduce more models that need to be run in parallel there is an increase in the resources needed to run the system.

Reliability:

2, 2	E1, E7:	Since the models are independent there is redundancy in the system, meaning that if one model becomes unavailable it does so by itself and the other models can continue operating as normal.

Maintainability:

3	E3:	Locating faults is much easier with smaller, more specialised, models rather than one big model.
−2	E3:	There are more models to maintain, and if we get new data all the models need to be retrained.

Portability:

−1	E1:	To port a system using parallel independent models, all the models would need to be compatible with the new hardware or software.

Model accuracy:

2	E7:	More channels of analysis allow for more information to be taken from the data.
2	E3:	Smaller, more specialised, models will allow each model to be more accurate within its limited scope than a larger, more general, model.

5.2.3 Sequentially Dependant Models

Reliability:

−2	E9:	Since the models are dependent on each other we have multiple locations where a problem would stop the whole system.

Maintainability:

−1,−1,−1	E2, E4, E9:	Since the models are dependent on each other we have multiple locations where a problem would stop the whole system.

Explainability:

1	E2:	If the outputs of each step are interpretable, the system will be easier to understand.
−2	E4:	It will become harder to mathematically explain where within the input data the prediction comes from since there will be more models and intermediary steps that the input goes through.

Model accuracy:

−1	E2:	Letting the model decide the steps itself usually results in better accuracy[2].

Fairness:

−1	E4:	As the data is processed many times over it will become more homogeneous, meaning that it will not represent outliers as well.

Compatibility:

2	E2:	Large models can be hard to get to interact with different software.Therefore, doing it in steps allows us to more easily change the models to interact with other software.

5.2.4 Situation-Based Model Selection

	E5:	*Highlighted that for this pattern to make sense the situation must be complex and require large amounts of data. In other cases, a model that handles all situations would probably be better overall.*

Resource efficiency:

2	E1:	As each model only operates in a specific area it can be smaller and require fewer resources to run.
2	E5:	Each model requires only a subset of the data to train on. This means training will be faster and less computationally expensive.
−1	E5:	More models will require more training.

Usability:

2	E7:	As we can choose different models for different scenarios we can choose the model that best suits the needs of the end user.

Maintainability:

2	E5:	As the models are independent they can be worked on independently.
1	E1:	As the functionality for different scenarios is divided between different models, locating the faults in the system will be easier.

Model accuracy:

1–2, 2	E1, E5:	As each model operates with a smaller input range it can be more specialised and its predictions simpler, providing more accurate predictions.

[2]The authors believe that this might be specific for larger models, detailed in section 6.4.

Fairness:

1	E1:	As each model operates within a known scenario it does not have to be biased by factors that do not need to be considered.

5.2.5 Data Transformation

Resource efficiency:

−2, −2	E4, E6:	As many steps transform the data, there will be a computational overhead.
2	E4:	As the functionality for different scenarios is divided between different models, locating the faults in the system will be easier.

Security:

1–2	E6:	When the data is processed in this way it will become unrecognisable from its original state and thus more secure.

Maintainability:

−2	E6:	As more steps in the processing of the data are added, there will be more code to maintain.

Portability:

−2 to −1	E6:	As this could need more computational power, porting it to a system requires the system to have higher performance.

Explainability:

−1	E4:	As the data is heavily processed it will be hard to trace the origins of a prediction.

Model accuracy:

2–3	E6:	As this is often a part of complex tasks, it is sometimes not even possible to get a functional model without this type of processing.
1	E4:	When data is heavily processed it tends to be more homogeneous, meaning that the mean prediction is going to be better.

Fairness:

−1	E4:	As we heavily process the data it tends to go towards the mean, meaning we will not represent outliers as well.

5.2.6 Feature Extraction

Resource efficiency:

?	E3:	Adding an extra step in the system will add some computational load.

Reliability:

2	E3:	Featurisation will allow the model to make better predictions outside of the range of the training data since we are only looking at the relevant features.
	E3:	*Highlighted that improper featurisation will make the model less reliable since it will not take all relevant factors into account.*

Security:

3, 2	E3, E6:	When featurising the data we can remove sensitive parts so that it becomes untraceable.

Explainability:

3	E3:	Since the score associated with a feature highlights its importance to the model, it will become easier to assign explanations for predictions.
-2 to -1	E6:	Highlighted that improper featurisation will make the model less reliable since it will not take all relevant factors into account.

Model accuracy:

2, 1–2	E3, E6:	When doing featurisation we allow the model to run on only the relevant features, meaning the prediction will only be based on the relevant features and thus be better.

Fairness:

3	E3:	When doing featurisation, we can hide biases in the data, allowing the model to be less biased.
	E3, E6:	*Highlighted that improper featurisation could introduce bias into the system, as it might remove a confounding factor that would mitigate the bias.*

5.2.7 Post Processing

Usability:

2	E10:	The step of post processing is inherently done to make the prediction from the model usable for the rest of the system and by extension the end user.
-2	E3:	In the process of doing post processing, data that could be useful for users might inadvertently be hidden.

Reliability:

?	E10:	During post processing, the output can be sanity-checked.

Fairness:

1	E10:	The classification thresholds can be changed in the post processing based on demographic attributes.

Compatibility:

2	E10:	Data can be post processed to be more usable by other software.

5.2.8 Prediction Cache

Resource efficiency:

2, 2	E4, E6:	When detecting duplicates the system does not have to perform unnecessary calculations, saving resources.

Reliability:

-2 to -1	E4:	For output uniqueness control, there is a chance for similar inputs to be classified as the same.

Security:

?, ?	E4, E6:	The data needs to be stored, bringing in security issues such as physical intrusion and SQL injections.

Portability:

-1	E6:	When porting this system the storage needs to be ported as well, which could make it more challenging.

5.2.9 Multi-Layer Pattern

Maintainability:

2–3, 3	E7, E9:	As the layers are separated they can be maintained independently.

Portability:

2–3	E7:	As layers are easier to change, hardware-specific layers can be swapped according to the present hardware.

5.2.10 Orchestrator

Resource efficiency:

3	E7:	The orchestrator can prioritize resources and balance loads in the runtime, which will improve the performance of the system.
-1	E8:	As each request has to go through the orchestrator and be routed, there will be a communication overhead.

Reliability:

-2	E7:	As all things connect to the orchestrator there is a single point of failure, meaning that if the orchestrator has problems the whole system will.

Security:

2	E8:	The orchestrator allows layering between the model and system, allowing for precautionary action and protection.

Maintainability:

3, 2	E7, E8:	The orchestrator separates concerns and allows the system to work with abstractions, meaning the underlying specifics can be changed without the system needing to change.
2	E8:	The orchestrator allows the models to be retrained more easily and on demand.

Portability:

1–2	E8:	Since the specifics of the model can more easily be changed, we can change the model depending on the available hardware.

Compatibility:

1	E8:	As we decouple the model, we allow other software to interact with abstractions, leading to easier communication.

5.2.11 Server-Side ML Model

Resource efficiency:

-1	E8:	Decoupling a system often brings communication overhead, slowing down the system slightly.

Security:

1–2, 2	E8, E9:	As we separate the model from the rest of the system we can place protections around the model.
?	E9:	Communication with the server can be intercepted.

Maintainability:

2–3	E8:	Separating the system and model allows the model to be changed more easily.

Portability:

2–3	E8:	When we can change the model more easily we can choose a model that can run on the sought hardware.

Compatibility:

1, 2	E8, E9:	Decoupling allows clients to interact with abstractions, leading to easier communication.

5.2.12 Expert Validation

Usability:

2	E5:	Since these systems are often safety critical the expert ensures the system is usable for an end user.

Reliability:

3	E1:	The expert can ensure that the output is correct, minimising incorrect predictions and increasing reliability.

Maintainability:

1–2	E5:	Expert can easily find if the system does not work as intended.

Fairness:

1	E5:	The expert can sanity-check the outputs for bias.
-1	E5:	The expert could introduce bias based on their previous experiences.
-1	E5:	A generative model could be more creative than the expert leading the expert to remove good answers, which would bring bias into the system..

5.2.13 Runtime Evaluation and Improvement

Reliability:

3, 3	E3, E10:	The model will give the best results when the input data is close to the training data. Therefore, constant monitoring and improvement of the model will ensure it is performing optimally even if the environment changes and the input data starts to drift from the original training data.

Maintainability:

-2 to -1	E3:	This type of system demands a good architecture since it needs to be able to change models autonomously.

Data quality:

-1	E3:	The newest data will always be the most valuable, but there will never be much of it. Therefore, this type of system might want to operate with that data which could lead to using a small amount of data in the system.

5.2.14 System Evaluation and Improvement

Resource efficiency:

2–3	E8:	As the system is monitored, decisions can be based on resource availability. Therefore, this type of system might want to operate with that data which could lead to using a small amount of data in the system.

Reliability:

2–3	E8:	The system can adapt if the metrics chosen for evaluation change, allowing the system to adapt to environmental changes.

Security:

2–3	E8:	The system can detect intrusions, allowing for active protection.

Maintainability:

-2	E9:	As the system changes, the evaluation technique might need to do so as well.

Compatibility:

-1	E8:	When the system can autonomously change it becomes harder for other software to rely on it. Since the system changing could bring issues to the other software relying on the system.

6

Discussion

In this chapter, we discuss the results by comparing them with related work and expand upon them. We also highlight some important aspects that might be of relevance to the results. Finally, we will discuss the various threats to the validity of the study, and how this research can be continued in the future.

6.1 Pattern Comparison

Between the now three literature reviews for finding patterns for AI or ML, only some of them have appeared in more than one piece of work, which is not unsurprising considering the difference in scope and search strategies between them. Not only can the time difference between the literature reviews play an impact in this rapidly evolving area of research, but also, this work only tries to identify patterns found in component models, while Heiland et al. searched for AI patterns, not ML specifically.

Table 6.1 lists the 14 identified patterns in this work, the 15 identified by Washizaki et al. [39], and the 25 architectural design patterns found by Heiland et al. [40], comparing which ones were found by the different literature reviews. Equivalent or similar patterns are placed on the same row. As you can see, there is zero overlap between this study's patterns and Washizaki et al's. Many of the patterns in their work, while still being general solutions to commonly occurring problems, can be seen as principles rather than patterns and are therefore not easy to recognise or visualise in component models, for example, *Discard proof of concept code* or *Data flows up, model flows down.*

This study	Heiland et al.	Washizaki et al.
Data transformation	Pipes and filters	
Ensemble learning		
Expert validation		
Feature extraction		
Multi-layer pattern	Multi-layer pattern	
Orchestrator	Mediator pattern	
Parallel independent models	Microservice horizontal pattern	
Post processing		
Prediction cache	Data cache / Prediction cache	
Runtime evaluation and improvement		
Sequentially dependant models	Microservice vertical pattern	
Server-side ML model	Client-server	
Situation-based model selection		
System evaluation and improvement		
	AI pipelines	
	Asynchronous pattern	
	Daisy architecture	
	Data lake	Data lake
	Data warehouse for ML	
	Distinguish business logic from ML model	Distinguish business logic from ML models
	Fluid architecture	
	Functional-style architecture	
	Handshake	
	Kappa architecture	Kappa architecture
	Lambda architecture	Lambda architecture for ML
	Model-View-Controller (MVC)	
	Parameter-server abstraction	
	Proxy pattern	
	Synchronous pattern	
	Tar pit	
	Workflow pipeline	
		Data flows up, model flows down
		Deployable canary model
		Different workloads in different computing environments
		Discard proof of concept code
		Encapsulate ML models within rule-base safeguards
		ML gateway routing architecture
		ML versioning
		Microservice architecture for ML
		Parameter-server abstraction
		Secure aggregation
		Separation of concerns and modularization of ML components

Table 6.1: Comparison of the overlap of patterns found in this work with the ones found by Heiland et al. [40] and the ones by Washizaki et al. [39].

6.2 Quality Attribute Impact Comparison

The results from the interviews, detailed in section 5.2, were insightful and provide many points where a pattern impacts the quality attributes of the system. However, because of the low number of interview time spent per pattern, the results may not be exhaustive.

Before the interviews took place, the authors systematically assessed each pattern and quality attribute combination to determine if they were connected inherently by the pattern's definition and the problem it solves. For example, it follows directly from the definition of expert validation, presented in section 5.1.4.1, that it is done to ensure the output of the system is correct, which falls under reliability. In this case, the effect is positive, but negative effects were also considered. By going through all combinations using this structured approach, a matrix was formed containing positive, negative, both positive and negative, as well as no or unknown impact for every combination of QA and pattern. Comparing this matrix with the results from the interview in Table 5.2, we get some discussion points, which will be discussed in this section.

These points are placed in the discussion as they are not a result, but rather something derived from the pattern definitions and the problems they solve, which are based on the authors' views. While some of these motivations are backed up by sources, further research must be done to confirm or disprove the other connections. Practitioners may consider these points but make their own judgement of their applicability to their system.

Each point below is marked with a minus (−) indicating that the connection is thought to have a negative impact, or a plus (+) indicating that the connection is thought to have a positive impact.

6.2.1 Ensemble Learning

Resource efficiency:

− To utilise ensemble learning there is a need to run multiple models, which would increase the resources needed for the system to get a prediction compared to running just one model.

6.2.2 Sequentially Dependant Models

Maintainability:

\+ Smaller more specialised models are easier to maintain than one large model.

6.2.3 Situation-Based Model Selection

Reliability:

+ As each model predicts based on a known situation its input space will be limited, meaning it will not have to worry about outliers in the same way that a combined model would have to.

6.2.4 Feature Extraction

Resource efficiency:

+ Extracting features allows the feature space to be small while keeping the information from the original feature space, which decreases the storage needed [58].

Maintainability:

+ Understanding the inner workings of the model will be easier when only working with the most relevant pieces of information.

6.2.5 Prediction Cache

Reliability:

– As the environment around a system changes the predictions could change as well, therefore a correct prediction stored at one time in the database might not be correct at a later date.

Maintainability:

– Adding a database to the system adds complexity to the system and another component to maintain.

– There is a need for policies for how data is stored and removed [40].

6.2.6 Multi-Layer Pattern

Compatibility:

+ Separating software into layers means that it is easier to build different layers with compatibility with other software in mind.

6.2.7 Server-Side ML Model

Reliability:

– Server-side ML model means that there is one point of failure.

Security:

+ Having the ML model in one remote place increases the confidentiality of the model.

Maintainability:

+ Maintenance of the model can now be done in one place whereas before, each instance of the model needed to be updated.

Portability:

+ If it is on a physical server this pattern enables you to detach the model from the system. Thus, the model does not have to run on the device that the system runs on, making the system more portable.

6.2.8 Expert Validation

Resource efficiency:

– The system would have to wait for human experts to interact with the system and do the validation. This will slow the system down since the validation will become a bottleneck and the system cannot run without the presence of a human expert.

Security:

– This pattern relies on a source outside the system that needs to interact with it, which could open up vulnerabilities that could be exploited.

6.2.9 Runtime Evaluation and Improvement

Resource efficiency:

– As we improve the model we will need to retrain it, which is a resource-intensive process.

6.3 Quality Attribute Impact Trends

Looking at Figure 5.2 and the discussion points in section 6.2, there are some interesting parallels we can see between the patterns. From the data, we can see that all the patterns categorised as system architecture patterns increase maintainability and portability (and if you consider the authors' discussion points, also compatibility). This makes sense for patterns about structuring the overall system architecture. They are concerned with how components interact within the system, allowing components to be more independent and more easily changed, increasing maintainability. This will in turn make portability increase since it will allow for changing of components depending on hardware or software needs. Since the interactions between components are also more clearly defined, interaction with other software will be easier, increasing compatibility.

It also seems that having multiple smaller more specialised models can be a strategy for increasing the system's model accuracy and reliability — unless we are talking about sequentially dependant models, which in contrast may decrease these quality attributes.

The experts agree that all validation patterns increase reliability, solidifying the name and purpose of this categorisation. Validating the input, output, the model, or even the system itself, seems to be a good way of increasing the reliability of the system.

Almost every pattern affects resource efficiency, reliability, and maintainability of the system in some way. If you want to increase data quality, though, then it seems you have to look for other options since none of the patterns impact data quality positively. The reason behind this could be that the data used for training the model is usually gathered outside of the system the model is located in — at least in the identified component models. Similarly, you may struggle to find patterns that will distinctly increase the explainability of the system. You may have to do work outside of the architecture of the system to find ways of increasing it.

Finally, only a few patterns are connected to the usability of the system. While the impact they have are positive, often it is not a strong impact. Because usability is connected to how the end-user perceives the system, which is more towards the user interface and user experience, it is difficult to quantify based on system architecture patterns. Therefore, usability is also a quality attribute that should be considered outside of the patterns used to build the system.

6.4 Impact of Model Size

In statistics, using models that have too many parameters is not recommended since it leads to overfitting and poor generalization [68]. But the current deep learning practices involve a great number of parameters, sometimes more parameters than data points. This is because of a phenomenon called *benign overfitting* [69] where the model overcomes the overfitting and achieves better accuracy than at the lowest point before the overfitting starts becoming a problem. It results in a so-called *double descent curve* as can be seen by the example in Figure 6.1, showing that the amount of errors decreases when you have a very large amount of parameters. The phenomenon was brought to the authors' attention by an interviewee when discussing the impact of model size on model accuracy after the interview concluded.

Since many of the interviewees for this thesis are researchers within ML, there is a chance some of their answers are based on this phenomenon, because research in ML often include state-of-the-art models that are very large. But as the term ML-enabled systems also encompass systems with smaller models, some of these answers might not be relevant for those systems. It could even be the case that those answers are misleading for systems with a smaller model size since implementing a model with a larger set of parameters then could increase overfitting and result in worse accuracy. This would mainly impact the model accuracy for the multiple model patterns since they break apart a bigger model into smaller ones.

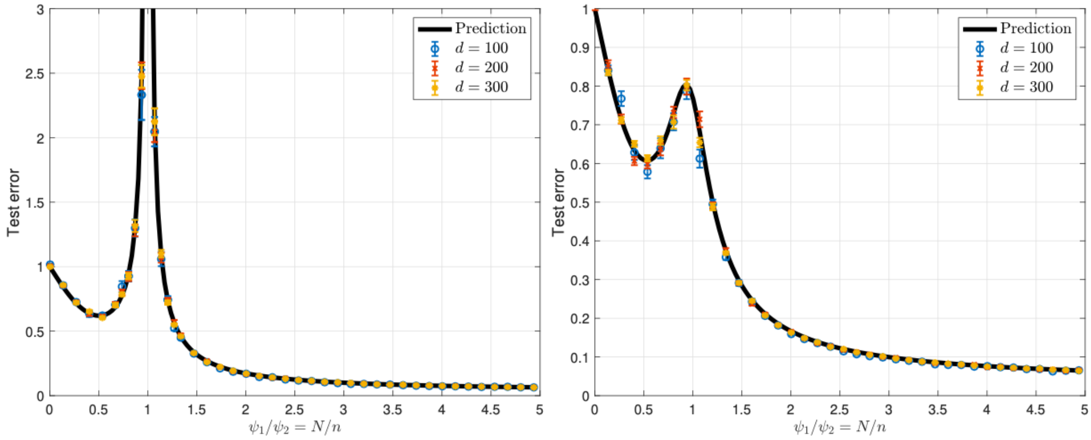

Figure 6.1: An example of *double descent curves* taken from [68].

6.5 Threats to Validity

The threats to the validity of a study directly influence its trustworthiness. Therefore, it is important to acknowledge where threats have been addressed, to strengthen the validity of the study, and where threat mitigation has been lacking, to be transparent on its weaknesses.

There are different approaches to categorise the threats to the validity of a study that is suitable for different types of studies [70]. The two most commonly used ones [71] are the one discussed by Runeson and Höst in their guidelines for case study research in software engineering [72], and the one discussed by Wohlin et al. in their book “Experimentation in Software Engineering” [73]. Both of these use construct, internal, and external validity. The former has an additional reliability while the latter instead uses conclusion validity.

This study is constructed by first performing a multivocal literature review, followed by pattern extraction, and finally interviews with domain experts. Thus, combining different categorisations of threats to validity is suitable for this study [70]. For the validity threats related to the literature review, i.e. the MLR performed to find component models of ML-enabled systems, Ampatzoglou et al.'s study “Identifying, categorizing and mitigating threats to validity in software engineering secondary studies” [71] will be used for threat categorisation and identification. Threats related to the remainder of the study will use the categorisation that Wohlin et al. discuss in [73].

The rest of the section is structured as follows: First, validity related to the performed multivocal literature review will be presented. After that, construct, internal, external, and conclusion validity related to the remainder of the study is discussed, i.e. the pattern extraction, interviews, and the study as a whole. Finally, a comment on the use of artificial intelligence as tools during the creation of this thesis.

6.5.1 Literature Review Validity

Ampatzoglou et al. highlights common issues related to reporting threats to validity in software engineering secondary studies [71]. One of these problems is that commonly reported threats are put in different validity categories, which the authors say is because of confusion within the community. To help bring back certainty, they propose a categorisation of threats related to secondary studies with an accompanied checklist of threat identifications and mitigations, which is why this is used in this study.

Below, study selection, data, and research validity threats related to the literature review are discussed as proposed by Ampatzoglou et al. [71]. These categories correspond to the planning phase, data extraction and analysis phase, and the overall design of the research, respectively.

6.5.1.1 Study Selection Validity

This section highlight validity threats related to the planning phase of the MLR, which includes the search process and filtering phases [71].

There is always a threat of not identifying all relevant studies when performing an SLR. In our case, the conditions are a bit different, as described in section 1.3, as well as the fact that identification of all grey literature sources is infeasible since there is such a large amount of them. Therefore, all relevant studies cannot be identified. However, we believe that an adequate number of them has been identified, both for this study's purposes and for potential further usage of the MLR.

We chose to not use meta databases, such as Google Scholar, since these databases search the primary databases we searched, and papers within the field of software engineering will most likely be published in one of the databases we searched. Another, more important, factor was that Google Scholar, which was the meta database we planned on using initially, did not support complex search queries. This would mean we would need to make multiple searches and manually apply the logical operations. Furthermore, Google Scholar does not enable exporting of search result and has good protection against web scraping, making this task infeasible.

Different variations of the search strings were piloted to ensure the ones used were as good as possible. Despite this, an oversight was made regarding the Google Image search string, thinking that parentheses were taken into account for queries when they in fact are ignored. The consequence of this, semantically speaking, was that some search results would contain "component diagram" without containing "machine learning", which means that this particular search would be less fruitful than its potential since a smaller number of sources could be included because of the study selection criteria. We say *semantically* because this was not something that was noticed in the result of the search, even after the error was found. It is believed by the authors that Google's algorithms are good enough to show quite relevant results despite this error. However, it is the case that the search string for Google Images was not optimal.

Interestingly, despite the suboptimal search string for Google Images, the amount of component models that were extracted from the search results were quite high. Comparing Figure 4.1 with Figure 4.2, we can see that Google Images had a large share of included sources compared to examined sources. In fact, the data from Table 4.1 shows that Google Images was the database with the highest individual source inclusion rate of 11%, with the total inclusion rate was 6.5%. There are two takeaways from this: First, this adds to the belief that Google's algorithms for ranking search results are quite good, with respect to the faulty logic of the used query. Second, the value grey literature can bring to a systematic literature review is evident.

In hindsight, the stopping criteria of 100 grey literature sources for Google Images was not optimal, because of the value it brought per checked source. It would instead have been optimal to keep going through results until you notice that the number of included sources starts to decrease. For instance, you could stop the search after 30 sources have been excluded in a row. This is an interpretation of *theoretical saturation* as described in Garousi et al.'s MLR guidelines [14], which was believed to not be suitable before the MLR was conducted. The overall effect this change would have had on the results is questionable, though it is impossible to know for sure. More patterns may have been found, but that does not take anything away from the patterns that we ended up with.

Another threat regarding the search string was the removal of sources containing "artificial intelligence". One of the reasons for doing this was that the number of sources was too big to be feasible for the given time frame since it would also snowball and slow down the later stages of the study, and excluding "artificial intelligence" removed roughly half of the sources. This did, however, mean that relevant sources might have been removed since for example this thesis contains the term "artificial intelligence" in its introduction, meaning it would not have appeared in the MLR search. While having fewer relevant sources is not optimal, the goal for the literature review was to find a collection of component models, and the authors believe that the 49 component models found were an adequate amount.

Further actions to mitigate threats to study selection validity were taken. The review protocol was reviewed by our supervisors before the MLR was started, which of course also helps with data validity. Additionally, the search results have been documented[1]. Two filtering phases were applied to effectivise work and help mitigate biases when applying the study selection criteria.

6.5.1.2 Data Validity

Data validity threats are about the data extraction and analysis phases of a secondary study [71]. How these threats relate to the performed MLR is discussed in this section.

Because the main data extracted was images of component models, the data ex-

[1]https://github.com/The-Impact-of-Design-Patterns-on-Quality-Attributes-in-ML-Enabled-Systems

traction process that was used was quite basic, meaning that most precautions to mitigate data extraction bias [71] is not applicable. While only one researcher did data extraction for a given source, the extracted component models were always verified by the other author afterwards.

The main data extraction for this study was not during the MLR, but after, when the component models had been extracted. Threats related to the pattern extraction that happened after the MLR was performed are discussed in section 6.5.3.

6.5.1.3 Research Validity

Threats regarding the overall research design of the MLR fall under the category of research validity [71] and will be discussed in this section.

Because of the selected definition of grey literature (section 2.2) and the way Google ranks the results[2], the MLR may not be repeatable in regards to the grey literature, even when using the same search strings. Although the sources used are documented you may get different results if you try to search again. This was a compensation we were willing to make for the sake of our study; performing an MLR was deemed the best way to answer our RQs.

The effect of this is that the results of the MLR — the list of component models of ML-enabled systems — may not be generalisable, coupled with the fact that no attempt was made to find all relevant primary sources. The reasons for the latter have already been established. However, the MLR is still generalisable to an extent and the component models could be used for further purposes, but the premises of how they were found needs to be taken into account.

6.5.2 Construct Validity

Construct validity is concerned with to what extent the measurements are suitable for the study and represents what is investigated [72], [73].

Evaluating design pattern impact on quality attributes is already established to be a challenging task [11], [32], [40], and after having performed it in this study, we would have to agree. The chosen methodology for the evaluation was expert opinion in the form of interviews, which induces threats to construct validity.

One interviewee highlighted an issue with the scale that was used during the interviews, seen in Figure 4.4. They were confused by what exactly is meant by a negative impact and would have rather expected two scales: Keep the one used but remove the negative part of the scale and have one additional scale for how important a quality attribute was, given a certain pattern. They argued that negative impacts do not make sense and instead what we are looking for is more so how critical it is to acknowledge and deal with. For example, they argued that reliability becomes more important when using the sequentially dependant models pattern since a whole line of ML models needs to be fault-tolerant and available. “More important” is neither a positive nor negative impact here according to them, just something that you have

[2]https://www.google.com/search/how-search-works/ranking-results/

to consider more extensively when designing the system. While having such scales could have been clearer and reduced confusion for some, the fact that it is two scales could also make it more confusing for others. From the authors' perspective, if an aspect requires more thought, time, and resources, it implies that there is a negative impact that needs to be mitigated. Since if a pattern is applied and this aspect is not given the consideration needed, the quality of the system will decrease.

If it is the fact that you have to go out of your way to make sure that a system implementing sequentially dependent models is reliable then its impact is negative. It is difficult to know for certain which scales are best to use for evaluating pattern impact, and whether or not the scale that was used measures the correct thing. Only one interviewee had issues with the scale as far as we know, however, we did not ask them about it. Which scale to use is a point that you have to carefully consider if similar work is done in the future.

6.5.3 Internal Validity

Internal validity threats are threats caused by causal relationships [72], [73]. In the case of our study, these threats can come about from subjectivity — even when an attempt is made to be objective — since if something is not fully objective it will always be prone to biases.

The pattern extraction process is the biggest threat to the internal validity of this study. While finding common patterns may seem to not contain subjectivity or biases at first glance, this is in fact not the case. The patterns are always created from the authors' interpretation of the component model, and even though this process was done thoroughly and systematically, it will never be completely free of subjectivity or biases. Actions to mitigate these threats were taken, e.g., the whole pattern extraction and creation process was done by both authors. Unfortunately, you can not get rid of, for example, anchoring bias in this way [74]. Furthermore, when a pattern was not found in related work, but rather coined by the authors, these are then formulated with unknown confounders that could affect the results. The problems a pattern solves may not be precise since they are written from the authors' understanding of what problems a certain pattern solves.

In fact, this internal validity threat is the reason the patterns' problems they solve were never mentioned during the interview. Thus, the interviewee only had the pattern name, definition, and an example to go on. Nevertheless, stating the reasons why a pattern is useful is necessary, otherwise, it is hard to know when to use it.

However, an underlying threat to the internal validity also lies in the pattern examples we showed during the interviews. Existing component models were used to highlight each pattern. Two exceptions were made for system evaluation and improvement and prediction cache since they had no corresponding clear component models. The examples that were chosen could have induced biases since the interviewees may have thought about the QA impact on the particular example system rather than a general implementation in any system.

For each interview, the patterns that were discussed were hand-picked for that expert. By tailoring the patterns to the expert in question, the thought was that it enables more valuable data to be extracted from the discussions, given that the person was more knowledgeable in the area that the pattern affects. For instance, the system architecture patterns were only shown to the experts from IDSE that will have a background in software engineering and be more familiar with software architecture.

6.5.4 External Validity

External validity refers to the generalisability of the study's findings [72], [73].

Because you could extract a component model from each system, even if not explicitly visualised, we could never achieve exhaustiveness in our literature review. Exhaustiveness can also not be guaranteed for the pattern extraction, since there could be patterns that were missed by the authors, notated in different ways, or just appeared once in this specific collection of component models (and thus not being considered a pattern in this study). However, as our study's main aim is to provide practitioners with some common patterns that can help with improving certain quality attributes when they are designing their ML-enabled system, this does not diminish the value of the study. This also ties in with the exhaustiveness of the connection and impact of patterns on the quality attributes, since if a practitioner is looking for a pattern to increase a certain quality attribute, it will be a pattern that heavily impacts the chosen quality attribute that will be chosen. Therefore, there is a bigger emphasis on covering the larger impacts of the patterns. Here, we are not looking for exhaustiveness either, as all patterns will in some way allow for impacts on all quality attributes. But, as these impacts are dependent on the implementation of the pattern we cannot say that these impacts are from the general pattern but rather the specific implementation of the pattern, which is not of use for the other practitioners. In conclusion, our study is not exhaustive in multiple ways, but importantly, this does not take away anything from its generalisability.

What does affect the generalisability of this study is the number of interviews. Because of a lack of responses from the people asked to participate in an interview, together with a lack of time to find more suitable participants, only ten interviews could be conducted. Since the purpose of this study is to provide practitioners with patterns and their associated quality attribute impacts, it was decided that an overview of the impact for all 14 patterns was more valuable than detailed impacts for a subset of them. This meant that we only had data from two or three interviewees per pattern. Of course, there are aspects of a pattern that may go unnoticed by two or three people (even if they are experts). This is why a discussion of points the experts may have missed is included in sections 6.2. However, from the data that was gathered, there was often a consensus between interviewee motivations, and seldom were they conflicting. Furthermore, since the most substantial data that was gathered is qualitative (the motivations for why a pattern impacts a certain quality attribute), there is still a lot of value despite having few interviewees. So, while our results regarding pattern impact may not be exhaustive, it is still believed to be very valuable when creating new ML-enabled systems because the results regarding reasons for a certain QA impact should still hold, thus making the results generalisable.

6.5.5 Conclusion Validity

The biggest threat to conclusion validity, which is the degree to which the conclusions reached are credible [73], is the fact that the authors of this study have limited experience with AI and ML. While both authors have some interest and general knowledge in AI and ML, it is more so the aspects regarding software architecture and quality that they are most comfortable with. This is relevant to highlight because of a point brought up by one interviewee regarding whether or not post processing and feature extraction are, in fact, patterns. They reasoned that it is used in basically all ML-enabled systems, and could not see how you can develop most systems without using these patterns. Therefore, they questioned the fact that they are considered as patterns and could not draw conclusions about their connection to QAs. Since they were found to be general solutions to occurring problems and depicted in component models, they were considered as patterns in this study. Whether or not they are obligatory for ML-enabled systems, which essentially means where to draw the line between what is part of the machine learning model and what is not. For example, is the model finished when the numerical representations of the answer have been produced, or when we have extracted the meaning behind the representations? This means that the step of extracting the meaning behind the representations could be seen as a post processing step, which means that the patterns mentioned might be present in all the systems, but some do not include them as separate components but rather as a part of the model. Therefore it cannot be stated by the authors whether “post processing” and “feature extraction” are really patterns or a part of all machine learning models.

6.5.6 Use of Artificial Intelligence

In the writing process of this thesis, generative AI has been utilised for improving the clarity and coherence of the texts. It is important to note that while answers from generative AI were utilised, they were never directly incorporated into the thesis, as they were instead viewed as suggestions on how to make the text better upon which we wrote our own text.

We also used the Google Recorder app for auto transcription of the interviews. These transcriptions were then checked manually since they contained some errors and extracting data from coherent text would be a lot easier. This step should therefore not impact the thesis, since errors were caught in the manual check, and only sped up our work.

6.6 Future Work

What is compelling with this study is that it can be easily expanded in different ways to broaden its results, especially since all material needed have been made publicly available in an online repository[3]. Using this material, more component models can be searched for and more patterns can be extracted from the found component models.

However, the most noteworthy point for future work is to hold more interviews to get further insight into the pattern impact on the associated quality attributes. This would add more validity to the results, and allow practitioners to be more certain about the impacts of the patterns. If this is done, it would be most beneficial to try to include experts within the field of software architecture for ML-enabled systems from industry. We believe that, since they would have knowledge about these systems in practice, they would add valuable insight that would make the study more applicable for the intended audience.

The alternative to using expert validation to answer RQ3 could have been to find an ML-enabled system without a specific pattern, implement the pattern, and quantitatively measure how it changes the system [32]. This method was considered as an alternative, but disregarded because it was viewed as having more threats to validity compared to expert validation for our particular study. For example, the change in the system will be dependent on the skill of the implementer, which should be left to people who have more experience with working with ML-enabled systems than the authors. Additionally, to achieve generalisability, this would also have to be done for multiple systems — a task that would have taken too long given our schedule. Furthermore, not all QAs can be evaluated in this way. Maintainability, model accuracy, and resource efficiency might be the only QAs that are easily quantifiable after implementing a pattern. However, a quantitative analysis of pattern impact on QAs would still be a good complement to expert opinions since it would be based on real systems and measurements.

[3]https://github.com/The-Impact-of-Design-Patterns-on-Quality-Attributes-in-ML-Enabled-Systems

7

Conclusion

In this thesis, a multivocal literature review was conducted that included 754 sources and resulted in a collection of 49 component models of ML-enabled systems. From these component models, 14 reoccurring architectural design patterns were found, which were presented in section 5.1. This collection of patterns answers RQ1: *What common patterns can be derived from existing component models of ML-enabled software systems in literature?*

These patterns were then presented to 10 domain experts in semi-structured interviews where they were asked to connect them to and rate their impact on quality attributes. From these answers Table 5.2 could be composed which answers RQ2: *What quality attributes are associated with the identified patterns?* Out of a possible 154 connections, 71 were made by the interviewees, which seems reasonable since not all patterns will impact all attributes. The detailed impact of the interview answers was then presented in section 5.2 which answers RQ3: *What impact do the patterns of the component models have on the quality attributes associated with them?*

This research will allow practitioners to more easily create ML-enabled systems with high quality. Because documentation of patterns in software has a positive impact on software comprehension and maintenance [11], finding commonly used patterns in ML-enabled systems allows for documentation of patterns within these systems, which was previously not possible to the same extent. This research also enables practitioners to, in the design phase, choose patterns depending on the quality attributes important to their system. They can do this since we have provided the correlated quality attribute impacts of each pattern, and when implementing the pattern they can expect the same quality attribute impacts [8]. But since the quality attribute impacts are compared to when the pattern is not used in a context where it could, a pattern implemented in the wrong context will not see the same quality attribute impact.

This thesis also laid a foundation for the research on the connection between architectural design patterns and quality attributes in ML-enabled systems. This then allows researchers within the field of software engineering for machine learning to expand upon and continue this research, increasing the much-needed knowledge about the quality of ML-enabled systems.

However, there are some limitations to this study, most notably the low number of interviewees. Since the number of interviewees was low, and the number of patterns high, the amount of data for each pattern could have been better. To reduce the effects of this, the authors provided their expanded view of the connections and impacts, provided in section 6.2. Some quality attributes can be inherently connected to quality attributes, and this section allows practitioners to read the motivations and decide if they agree with the inherency claimed by the authors. This is the clearest point for future work and to interview more experts to get more detailed and robust connections and impacts.

Even with this downside, the authors believe that the research results are promising and can be used beneficially. The collection of patterns is insightful and the quality attribute impact is a valuable addition, differentiating this research from the previous work on architectural design patterns within ML-enabled systems. This key difference will allow this research to be more directly applicable for practitioners, leading to higher overall quality of ML-enabled systems.

Bibliography

[1] D. Sculley, G. Holt, D. Golovin, *et al.*, "Hidden technical debt in machine learning systems," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 2503–2511.

[2] G. A. Lewis, I. Ozkaya, and X. Xu, "Software architecture challenges for ml systems," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 634–638. DOI: 10.1109/ICSME52107.2021.00071.

[3] I. Stoica, D. Song, R. A. Popa, *et al.*, "A berkeley view of systems challenges for ai," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-159, Oct. 2017. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-159.html.

[4] H. Muccini and K. Vaidhyanathan, *Software architecture for ml-based systems: What exists and what lies ahead*, 2021. arXiv: 2103.07950 [cs.SE].

[5] F. Kumeno, "Sofware engneering challenges for machine learning applications: A literature review," *Intelligent Decision Technologies*, vol. 13, pp. 463–476, Feb. 2020. DOI: 10.3233/IDT-190160.

[6] K. R. Thórisson, "Integrated a.i. systems," *Minds Mach.*, vol. 17, no. 1, pp. 11–25, Mar. 2007, ISSN: 0924-6495. DOI: 10.1007/s11023-007-9055-5. [Online]. Available: https://doi.org/10.1007/s11023-007-9055-5.

[7] U. S. D. of Defence. "Component models." (), [Online]. Available: https://dodcio.defense.gov/Portals/0/Documents/DODAF/Vol_1_Sect_7-2-1_Component_Models.pdf (visited on 03/01/2024).

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice.* Addison-Wesley Professional, 2012, ISBN: 978-0-321-81573-6.

[9] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994, ISBN: 0201633612.

[10] P. Bourque, R. E. Fairley, and I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd. Washington, DC, USA: IEEE Computer Society Press, 2014, ISBN: 0769551661.

[11] F. Wedyan and S. Abufakher, "Impact of design patterns on software quality: A systematic literature review," *IET Software*, vol. 14, no. 1, pp. 1–17, 2020. DOI: https://doi.org/10.1049/iet-sen.2018.5446. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.

2018.5446. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5446.

[12] N. Maslej, L. Fattorini, E. Brynjolfsson, *et al.*, *Artificial intelligence index report 2023*, 2023. arXiv: 2310.03715 [cs.AI].

[13] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, Jan. 2007.

[14] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, pp. 101–121, 2019, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.09.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918301939.

[15] V. Garousi, M. Felderer, and M. V. Mäntylä, "The need for multivocal literature reviews in software engineering," 2016. DOI: 10.1145/2915970.2916008.

[16] G. T. G. Neto, W. B. Santos, P. T. Endo, and R. A. Fagundes, "Multivocal literature reviews in software engineering: Preliminary findings from a tertiary study," 2019. DOI: 10.1109/esem.2019.8870142.

[17] H. Cooper, L. Hedges, and J. Valentine, "The handbook of research synthesis and meta-analysis 2nd edition," English (US), in *The Hand. of Res. Synthesis and Meta-Analysis, 2nd Ed.* Russell Sage Foundation, Dec. 2009, pp. 1–615, ISBN: 9780871541635.

[18] J. Schöpfel, "Towards a Prague Definition of Grey Literature," in *Twelfth International Conference on Grey Literature: Transparency in Grey Literature. Grey Tech Approaches to High Tech Issues. Prague, 6-7 December 2010*, Czech Republic, Dec. 2010, pp. 11–26. [Online]. Available: https://archivesic.ccsd.cnrs.fr/sic_00581570.

[19] R. J. Adams, P. Smart, and A. S. Huff, "Shades of grey: Guidelines for working with the grey literature in systematic reviews for management and organizational studies," *International Journal of Management Reviews*, vol. 19, no. 4, pp. 432–454, 2017. DOI: https://doi.org/10.1111/ijmr.12102. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ijmr.12102. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/ijmr.12102.

[20] B. Kitchenham, L. Madeyski, and D. Budgen, "How should software engineering secondary studies include grey material?" *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 872–882, 2023. DOI: 10.1109/TSE.2022.3165938.

[21] V. Garousi, M. Felderer, M. V. Mäntylä, and A. Rainer, "Benefitting from the grey literature in software engineering research," in *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Cham: Springer International Publishing, 2020, pp. 385–413, ISBN: 978-3-030-32489-6. DOI: 10.1007/978-3-030-32489-6_14. [Online]. Available: https://doi.org/10.1007/978-3-030-32489-6_14.

[22] International Organization for Standardization (ISO), "System and software quality models," ISO, International Standard ISO/IEC 25010:2011(E), 2011.

[Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.

[23] SEBoK. "Quality management." Q. Wang, M. Towhidnejad, and D. Olwell, Eds. version 2.9. Accessed: 2024-03-05. (Nov. 20, 2023), [Online]. Available: https://sebokwiki.org/wiki/Quality_Management.

[24] V. Indykov, D. Strüber, and R. Wohlrab, "Architectural tactics to achieve common quality attributes of machine-learning-enabled systems," In submission, 2023.

[25] International Organization for Standardization, "Quality model for ai systems," ISO, International Standard ISO/IEC 25059:2023, 2023. [Online]. Available: https://www.iso.org/standard/80655.html.

[26] F. P. B. Jr, *The Mythical Man-Month, Essays on software engineering.* Addison-Wesley, 1972, ISBN: 0-201-00650-2.

[27] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992, ISSN: 0163-5948. DOI: 10.1145/141874.141884. [Online]. Available: https://doi.org/10.1145/141874.141884.

[28] D. Garlan and M. Shaw, "An introduction to software architecture," USA, Tech. Rep., 1994.

[29] E. Woods and N. Rozanski, "Using architectural perspectives," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 2005, pp. 25–35. DOI: 10.1109/WICSA.2005.74.

[30] E. Woods, "Software architecture in a changing world," *IEEE Software*, vol. 33, no. 6, pp. 94–97, 2016. DOI: 10.1109/MS.2016.149.

[31] A. Vogelsang and M. Borg, "Requirements engineering for machine learning: Perspectives from data scientists," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, 2019, pp. 245–251. DOI: 10.1109/REW.2019.00050.

[32] B. Bafandeh Mayvan, A. Rasoolzadegan, and Z. Ghavidel Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *Journal of Systems and Software*, vol. 125, pp. 93–118, 2017, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.11.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216302321.

[33] Q. Lu, L. Zhu, X. Xu, J. Whittle, D. Zowghi, and A. Jacquet, *Responsible ai pattern catalogue: A collection of best practices for ai governance and engineering*, 2023. arXiv: 2209.04963 [cs.AI].

[34] J. Juziuk, D. Weyns, and T. Holvoet, "Design patterns for multi-agent systems: A systematic literature review," in *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, O. Shehory and A. Sturm, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 79–99, ISBN: 978-3-642-54432-3. DOI: 10.1007/978-3-642-54432-3_5. [Online]. Available: https://doi.org/10.1007/978-3-642-54432-3_5.

[35] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez, and N. Yoshioka, "Landscape of architecture and design patterns for iot systems,"

IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10 091–10 101, 2020. DOI: 10.1109/JIOT.2020.3003528.

[36] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," Mar. 2018. DOI: 10.5220/0006798302210232.

[37] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. Ocharán-Hernández, "Patterns related to microservice architecture: A multivocal literature review," *Programming and Computer Software*, vol. 46, no. 8, pp. 594–608, 2020, ISSN: 0361-7688. DOI: 10.1134/s0361768820080253.

[38] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc, "Studying software engineering patterns for designing machine learning systems," in *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, 2019, pp. 49–495. DOI: 10.1109/IWESEP49350.2019.00017.

[39] H. Washizaki, F. Khomh, Y. Gueheneuc, *et al.*, "Software-engineering design patterns for machine learning applications," *Computer*, vol. 55, no. 03, pp. 30–39, Mar. 2022, ISSN: 1558-0814. DOI: 10.1109/MC.2021.3137227.

[40] L. Heiland, M. Hauser, and J. Bogner, *Design patterns for ai-based systems: A multivocal literature review and pattern repository*, 2023. arXiv: 2303.13173 [cs.SE].

[41] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns.* Wiley Publishing, 1996, ISBN: 0471958697.

[42] G. Márquez and H. Astudillo, "Actual use of architectural patterns in microservices-based open source projects," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 31–40. DOI: 10.1109/APSEC.2018.00017.

[43] M. Rahman, M. S. H. Chy, and S. Saha, "A systematic review on software design patterns in today's perspective," in *2023 IEEE 11th International Conference on Serious Games and Applications for Health (SeGAH)*, 2023, pp. 1–8. DOI: 10.1109/SeGAH57547.2023.10253758.

[44] E. Knauss, *Constructive master's thesis work in industry: Guidelines for applying design science research*, 2021. arXiv: 2012.04966 [cs.SE].

[45] Google. "Why your google search results might differ from other people." (2024), [Online]. Available: https://support.google.com/websearch/answer/12412910 (visited on 02/20/2024).

[46] Y. Wu, P. Gupta, M. Wei, Y. Acar, S. Fahl, and B. Ur, "Your secrets are safe," 2018. DOI: 10.1145/3178876.3186088.

[47] S. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *11th IEEE International Software Metrics Symposium (METRICS'05)*, 2005, 10 pp.–23. DOI: 10.1109/METRICS.2005.24.

[48] J. W. Drisko and T. Maschi, *Content analysis.* Oxford University Press, USA, 2016.

[49] G. Terry, N. Hayfield, V. Clarke, V. Braun, *et al.*, "Thematic analysis," *The SAGE handbook of qualitative research in psychology*, vol. 2, no. 17-37, p. 25, 2017.

[50] A. Singh. "Comprehensive guide for ensemble models." (2023), [Online]. Available: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/ (visited on 04/11/2024).

[51] J. Brownlee. "A tour of ensemble learning algorithms." (2021), [Online]. Available: https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/ (visited on 04/10/2024).

[52] X. H. Vu, X. D. Hoang, and T. H. H. Chu, "A novel model based on ensemble learning for detecting dga botnets," in *2022 14th International Conference on Knowledge and Systems Engineering (KSE)*, 2022, pp. 1–6. DOI: 10.1109/KSE56063.2022.9953792.

[53] Z. Peng, J. Yang, T.-H. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems: A case study on apollo," ser. ESEC/FSE 2020, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1240–1250, ISBN: 9781450370431. DOI: 10.1145/3368089.3417063. [Online]. Available: https://doi.org/10.1145/3368089.3417063.

[54] M. Abirami, R. Anand, and A. M. Bhaskaran, "Facial image-based age and gender classification to enhance recommendations in a smart tv environment," in *2023 International Conference on Networking and Communications (ICNWC)*, 2023, pp. 1–7. DOI: 10.1109/ICNWC57852.2023.10127283.

[55] S. Kumar. "Strategy pattern | set 1 (introduction)." (2024), [Online]. Available: https://www.geeksforgeeks.org/strategy-pattern-set-1/ (visited on 04/11/2024).

[56] S. M. Anjum, K. M. Malik, H. Soltanian-Zadeh, H. Malik, and G. Malik, "Saccular brain aneurysm detection and multiclassifier rupture prediction using digital subtraction and magnetic resonance angiograms," in *Proceedings of the 2018 5th International Conference on Biomedical and Bioinformatics Engineering*, ser. ICBBE '18, Okinawa, Japan: Association for Computing Machinery, 2018, pp. 87–94, ISBN: 9781450365611. DOI: 10.1145/3301879.3301899. [Online]. Available: https://doi.org/10.1145/3301879.3301899.

[57] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, *Feature Extraction: Foundations and Applications* (Studies in Fuzziness and Soft Computing). Springer Berlin Heidelberg, 2008, ISBN: 9783540354888. [Online]. Available: https://books.google.se/books?id=FOTzBwAAQBAJ.

[58] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 Science and Information Conference*, 2014, pp. 372–378. DOI: 10.1109/SAI.2014.6918213.

[59] A. D. Jain and C. M. Hera, "Noise reduction using machine learning," 2018. [Online]. Available: https://patents.google.com/patent/US20190122689A1/en (visited on 03/07/2024).

[60] I. Awaad and B. León, "Xpersif: A software integration framework and architecture for robotic learning by experimentation," Ph.D. dissertation, Feb. 2008.

[61] P. Laplante, *What Every Engineer Should Know about Software Engineering* (What Every Engineer Should Know). CRC Press, 2007, pp. 85–90, ISBN:

9781420006742. [Online]. Available: https://books.google.se/books?id=pFHYk0KWAEgC.

[62] F. Petroni, N. Raman, T. Nugent, *et al.*, "An extensible event extraction system with cross-media event resolution," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18, London, United Kingdom: Association for Computing Machinery, 2018, pp. 626–635, ISBN: 9781450355520. DOI: 10.1145/3219819.3219827. [Online]. Available: https://doi.org/10.1145/3219819.3219827.

[63] L. Dorard. "9 components of a real-world machine learning system." (2020), [Online]. Available: https://www.linkedin.com/pulse/9-components-real-world-machine-learning-system-louis-dorard/ (visited on 03/07/2024).

[64] Y. Zi, Y. Luo, Z. Guang, L. Qi, T. Wu, and X. Zhang, "Anomalous taxi route detection system based on cloud services," in *Cloud Computing, Smart Grid and Innovative Frontiers in Telecommunications*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 322, 2020, pp. 240–254. DOI: 10.1007/978-3-030-48513-9_20. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-48513-9_20.

[65] Commonwealth Scientific and Industrial Research Organisation (CSIRO). "Multi-model decision maker." (2023), [Online]. Available: https://research.csiro.au/ss/science/projects/responsible-ai-pattern-catalogue/multi-model-decision-maker/ (visited on 03/07/2024).

[66] Amazon Web Services. "ML lifecycle architecture diagram." (2024), [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/ml-lifecycle-architecture-diagram.html (visited on 03/07/2024).

[67] S. Sheikhi and S. M. Babamir, "A predictive framework for load balancing clustered web servers," *Journal of Supercomputing*, vol. 72, pp. 588–611, 2016. DOI: 10.1007/s11227-015-1584-8. [Online]. Available: https://doi.org/10.1007/s11227-015-1584-8.

[68] S. Mei and A. Montanari, "The generalization error of random features regression: Precise asymptotics and the double descent curve," *Communications on Pure and Applied Mathematics*, vol. 75, no. 4, pp. 667–766, 2022. DOI: https://doi.org/10.1002/cpa.22008. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.22008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.22008.

[69] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler, "Benign overfitting in linear regression," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 117, no. 48, pp. 30 063–30 070, Dec. 2020, Epub 2020 Apr 24, ISSN: 1091-6490. DOI: 10.1073/pnas.1907378117. eprint: https://www.pnas.org/content/117/48/30063.full.pdf. [Online]. Available: https://www.pnas.org/content/117/48/30063.

[70] R. Verdecchia, E. Engström, P. Lago, P. Runeson, and Q. Song, "Threats to validity in software engineering research: A critical reflection," *Information and Software Technology*, vol. 164, p. 107 329, 2023, ISSN: 0950-5849.

DOI: https://doi.org/10.1016/j.infsof.2023.107329. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584923001842.

[71] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies," *Information and Software Technology*, vol. 106, pp. 201–230, 2019, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.10.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918302106.

[72] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," English, *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8.

[73] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[74] B. Fischhoff, D. Kahneman, P. Slovic, and A. Tversky, "Debiasing," *Judgment under uncertainty: Heuristics and biases*, Jan. 1982.

[75] X. Wu and V. Taylor, "Utilizing ensemble learning for performance and power modeling and improvement of parallel cancer deep learning candle benchmarks," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 15, e6516, 2023. DOI: https://doi.org/10.1002/cpe.6516. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6516. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6516.

[76] A. Tufek and M. S. Aktas, "On the provenance extraction techniques from large scale log files," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 15, e6559, 2023. DOI: https://doi.org/10.1002/cpe.6559. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6559. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6559.

[77] N. Gatto, E. Kusmenko, and B. Rumpe, "Modeling deep reinforcement learning based architectures for cyber-physical systems," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 196–202. DOI: 10.1109/MODELS-C.2019.00033.

[78] H. Zhang, L. Kuang, A. Wu, Q. Zhao, and X. Yang, "Just-in-time defect prediction enhanced by the joint method of line label fusion and file filtering," *IET Software*, vol. 17, no. 4, pp. 378–391, 2023. DOI: https://doi.org/10.1049/sfw2.12131. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/sfw2.12131. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/sfw2.12131.

[79] P. Nawrocki and B. Sniezynski, "Adaptive service management in mobile cloud computing by means of supervised and reinforcement learning," *Journal of Network and Systems Management*, vol. 26, no. 1, pp. 1–22, 2018, ISSN:

1064-7570. DOI: 10.1007/s10922-017-9405-4. [Online]. Available: https://dx.doi.org/10.1007/s10922-017-9405-4.

[80] S. Imai, S. Chen, W. Zhu, and C. A. Varela, "Dynamic data-driven learning for self-healing avionics," *Cluster Computing*, vol. 22, no. S1, pp. 2187–2210, 2019, ISSN: 1386-7857. DOI: 10.1007/s10586-017-1291-8.

[81] N. Shafi, M. Abdullah, W. Iqbal, A. Erradi, and F. Bukhari, "Cdascaler: A cost-effective dynamic autoscaling approach for containerized microservices," *Cluster Computing*, 2024, ISSN: 1386-7857. DOI: 10.1007/s10586-023-04228-y.

[82] M. Gravina, S. Marrone, G. Piantadosi, V. Moscato, and C. Sansone, "Developing a smart pacs: Cbir system using deep learning," in *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, 2021, pp. 296–309. DOI: 10.1007/978-3-030-68790-8_24.

[83] M. Labiadh, C. Obrecht, C. Ferreira Da Silva, and P. Ghodous, "A microservice-based framework for exploring data selection in cross-building knowledge transfer," *Service Oriented Computing and Applications*, vol. 15, no. 2, pp. 97–107, 2021, ISSN: 1863-2386. DOI: 10.1007/s11761-020-00306-w.

[84] Y. Touzani and K. Douzi, "An lstm and gru based trading strategy adapted to the moroccan market," *Journal of Big Data*, vol. 8, no. 1, 2021, ISSN: 2196-1115. DOI: 10.1186/s40537-021-00512-z.

[85] S. Mittal and J. L. Risco-Martín, "Simulation-based complex adaptive systems," in *Simulation Foundations, Methods and Applications*. Simulation Foundations, Methods and Applications, 2017, pp. 127–150. DOI: 10.1007/978-3-319-61264-5_6.

[86] L. Bortolussi, D. Milios, and G. Sanguinetti, "U-check: Model checking and parameter synthesis under uncertainty," in *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, 2015, pp. 89–104. DOI: 10.1007/978-3-319-22264-6_6.

[87] J. Singh, G. Goyal, and S. Gupta, "Fadu-ev an automated framework for pre-release emotive analysis of theatrical trailers," *Multimedia Tools and Applications*, vol. 78, no. 6, pp. 7207–7224, 2019, ISSN: 1380-7501. DOI: 10.1007/s11042-018-6412-8.

[88] I. Verenich, S. Mõškovski, S. Raboczi, M. Dumas, M. La Rosa, and F. M. Maggi, "Predictive process monitoring in apromore," in *Lecture Notes in Business Information Processing*. Lecture Notes in Business Information Processing, 2018, pp. 244–253. DOI: 10.1007/978-3-319-92901-9_21.

[89] T. Borangiu, E. V. Oltean, S. Răileanu, I. Iacob, S. Anton, and F. Anton, "Modelling service processes as discrete event systems with arti-type holonic control architecture," in *Lecture Notes in Business Information Processing*. Lecture Notes in Business Information Processing, 2020, pp. 377–390. DOI: 10.1007/978-3-030-38724-2_27.

[90] S. Wang, Y. Zhong, H. Lu, E. Wang, W. Yun, and W. Cai, "Geospatial big data analytics engine for spark," in *Proceedings of the 6th ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial '17, Redondo Beach, CA, USA: Association for Computing Machinery, 2017, pp. 42–45,

ISBN: 9781450354943. DOI: 10.1145/3150919.3150923. [Online]. Available: https://doi.org/10.1145/3150919.3150923.

[91] M. Guerriero, D. A. Tamburri, Y. Ridene, F. Marconi, M. M. Bersani, and M. Artac, "Towards devops for privacy-by-design in data-intensive applications: A research roadmap," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ser. ICPE '17 Companion, L'Aquila, Italy: Association for Computing Machinery, 2017, pp. 139–144, ISBN: 9781450348997. DOI: 10.1145/3053600.3053631. [Online]. Available: https://doi.org/10.1145/3053600.3053631.

[92] Y. Zhang and A. Kumar, "Panorama: A data system for unbounded vocabulary querying over video," *Proc. VLDB Endow.*, vol. 13, no. 4, pp. 477–491, Dec. 2019, ISSN: 2150-8097. DOI: 10.14778/3372716.3372721. [Online]. Available: https://doi.org/10.14778/3372716.3372721.

[93] J. F. Perez, W. Wang, and G. Casale, "Towards a devops approach for software quality engineering," in *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*, ser. WOSP '15, Austin, Texas, USA: Association for Computing Machinery, 2015, pp. 5–10, ISBN: 9781450333405. DOI: 10.1145/2693561.2693564. [Online]. Available: https://doi.org/10.1145/2693561.2693564.

[94] J. Deng, W. Wang, P. Venugopal, J. Popovic, and G. Rietveld, "Knowledge-aware artificial neural network for loss modeling of planar magnetic components," in *2022 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2022, pp. 1–6. DOI: 10.1109/ECCE50734.2022.9947398.

[95] S. Jiang and M. Shen, "A cognitive urban collision avoidance framework based on agents priority using recurrent neural network," in *2019 19th International Conference on Advanced Robotics (ICAR)*, 2019, pp. 474–480. DOI: 10.1109/ICAR46387.2019.8981566.

[96] C. Yu, R. E. Zezario, S.-S. Wang, *et al.*, "Speech enhancement based on denoising autoencoder with multi-branched encoders," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2756–2769, 2020. DOI: 10.1109/TASLP.2020.3025638.

[97] V. A. Yemelyanov, A. R. Fatkulin, A. A. Nedelkin, V. A. Titov, and A. V. Degtyarev, "Software for weight estimation of the transported liquid iron," in *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 2019, pp. 381–384. DOI: 10.1109/EIConRus.2019.8657011.

[98] M. Razian, M. Fathian, H. Wu, A. Akbari, and R. Buyya, "Saiot: Scalable anomaly-aware services composition in cloudiot environments," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3665–3677, 2021. DOI: 10.1109/JIOT.2020.3023938.

[99] I. García-Magariño, M. Nasralla, and S. Nazir, "Real-time analysis of online sources for supporting business intelligence illustrated with bitcoin investments and iot smart-meter sensors in smart cities," *Electronics*, vol. 9, p. 1101, Jul. 2020. DOI: 10.3390/electronics9071101.

[100] M. Schmitt. "Machine learning architecture: The core components." (2022), [Online]. Available: https://www.datarevenue.com/en-blog/machine-learning-project-architecture (visited on 06/10/2024).

[101] X. Shan, T. Huang, L. Luo, *et al.*, "Automatic recognition of microstructures of air-plasma-sprayed thermal barrier coatings using a deep convolutional neural network," *Coatings*, vol. 13, Dec. 2022. DOI: 10.3390/coatings13010029.

[102] C. Kästner. "Machine learning in production: From models to systems." (2022), [Online]. Available: https://ckaestne.medium.com/machine-learning-in-production-from-models-to-systems-e1422ec7cd65 (visited on 06/10/2024).

[103] A. A. M. Thakur, M. Zambito, R. K. Archibald, *et al.* "Machine learning in production: From models to systems." (2022), [Online]. Available: https://www.ornl.gov/research-highlight/integrating-machine-learning-microscope-control-using-intersect (visited on 06/10/2024).

[104] S. Imai, S. Chen, W. Zhu, and C. A. Varela, "Dynamic data-driven learning for self-healing avionics," Nov. 2017, ISSN: 1573-7543. DOI: 10.1007/s10586-017-1291-8. [Online]. Available: http://rdcu.be/yJNh.

[105] F. T. J. A. Carvajal Soto and D. Gyulai, "An online machine learning framework for early detection of product failures in an industry 4.0 context," *International Journal of Computer Integrated Manufacturing*, vol. 32, no. 4-5, pp. 452–465, 2019. DOI: 10.1080/0951192X.2019.1571238. [Online]. Available: https://doi.org/10.1080/0951192X.2019.1571238.

[106] T. Nishi, "Online learning and vehicle control method based on reinforcement learning without active exploration," 2016. [Online]. Available: https://patents.google.com/patent/US10065654B2/en (visited on 06/10/2024).

[107] B. Englard, G. Gandhi, and P. Maheshwari, "Controlling an autonomous vehicle using cost maps," 2018. [Online]. Available: https://patents.google.com/patent/US10606270B2/en (visited on 06/10/2024).

[108] X. Liu, H. Cheng, Y. Wang, *et al.*, "Adversarial pretraining of machine learning models," 2020. [Online]. Available: https://patents.google.com/patent/US20210326751A1/en (visited on 06/10/2024).

[109] L. Mercep and M. Pollach, "Embedded automotive perception with machine learning classification of sensor data," 2018. [Online]. Available: https://patents.google.com/patent/US20180314253A1/en (visited on 06/10/2024).

[110] T. N. Blair, A. A. Kurzhanskiy, S. J. Lazaris, *et al.*, "Sentiment and rules-based equity analysis using customized neural networks in multi-layer, machine learning-based model," 2020. [Online]. Available: https://patents.google.com/patent/US20220036461A1/en (visited on 06/10/2024).

[111] C. Lan, W. Zeng, S. SONG, and J. Xing, "Neural network-based action detection," 2016. [Online]. Available: https://patents.google.com/patent/US11003949B2/en (visited on 06/10/2024).

A

Multivocal Literature Review Component Models

ID	Database	Source title	Figure	Reference
W1	Wiley	Utilizing ensemble learning for performance and power modeling and improvement of parallel cancer deep learning CANDLE benchmarks	1	[75]
W5	Wiley	On the provenance extraction techniques from large scale log files	2	[76]
W9	IEEE Xplore	Modeling Deep Reinforcement Learning Based Architectures for Cyber-Physical Systems	1	[77]
W10	Wiley	Just-in-time defect prediction enhanced by the joint method of line label fusion and file filtering	4	[78]
W24	SpringerLink	Adaptive Service Management in Mobile Cloud Computing by Means of Supervised and Reinforcement Learning	1	[79]
W30	SpringerLink	Dynamic data-driven learning for self-healing avionics	3	[80]
W31	SpringerLink	Cdascaler: a cost-effective dynamic autoscaling approach for containerized microservices	2	[81]
W35	ACM DL	A first look at the integration of machine learning models in complex autonomous driving systems: a case study on Apollo	1	[53]
W37	SpringerLink	Developing a Smart PACS: CBIR System Using Deep Learning	1	[82]
W49	SpringerLink	A microservice-based framework for exploring data selection in cross-building knowledge transfer	3	[83]
W53	SpringerLink	An LSTM and GRU based trading strategy adapted to the Moroccan market	3	[84]
W55	SpringerLink	Simulation-Based Complex Adaptive Systems	6.1	[85]
W95	SpringerLink	A predictive framework for load balancing clustered web servers	2	[67]
W120	SpringerLink	Anomalous Taxi Route Detection System Based on Cloud Services	2	[64]
W155	ACM DL	An Extensible Event Extraction System With Cross-Media Event Resolution	1	[62]
W167	ACM DL	Saccular Brain Aneurysm Detection and Multiclassifier Rupture Prediction using Digital Subtraction and Magnetic Resonance Angiograms	2	[56]
W201	SpringerLink	U-Check: Model Checking and Parameter Synthesis Under Uncertainty	1	[86]

ID	Database	Source title	Figure	Reference
W217	SpringerLink	FADU-EV an automated framework for pre-release emotive analysis of theatrical trailers	1	[87]
W232	SpringerLink	Predictive Process Monitoring in Apromore	1	[88]
W259	SpringerLink	Modelling Service Processes as Discrete Event Systems with ARTI-Type Holonic Control Architecture	3	[89]
W380	ACM DL	Geospatial Big Data Analytics Engine for Spark	3	[90]
W394	ACM DL	Towards DevOps for Privacy-by-Design in Data-Intensive Applications: A Research Roadmap	2	[91]
W405	ACM DL	Panorama: a data system for unbounded vocabulary querying over video	5	[92]
W442	ACM DL	Towards a DevOps Approach for Software Quality Engineering	1	[93]
W469	IEEE Xplore	Facial Image-based Age and Gender Classification to Enhance Recommendations in a Smart TV Environment	3	[54]
W481	IEEE Xplore	Knowledge-aware Artificial Neural Network for Loss Modeling of Planar Magnetic Components	4	[94]
W491	IEEE Xplore	A Cognitive Urban Collision Avoidance Framework Based on Agents Priority Using Recurrent Neural Network	2	[95]
W500	IEEE Xplore	A Novel Model Based on Ensemble Learning for Detecting DGA Botnets	2	[52]
W502	IEEE Xplore	Speech Enhancement Based on Denoising Autoencoder With Multi-Branched Encoders	1	[96]
W518	IEEE Xplore	Software for Weight Estimation of the Transported Liquid Iron	3	[97]
W537	IEEE Xplore	SAIoT: Scalable Anomaly-Aware Services Composition in CloudIoT Environments	2	[98]
G6	Google Images	The 9 components in a real-world Machine Learning system	1	[63]
G11	Google Images	Real-Time Analysis of Online Sources for Supporting Business Intelligence Illustrated with Bitcoin Investments and IoT Smart-Meter Sensors in Smart Cities	2	[99]
G12	Google Images	Machine Learning Architecture: The Core Components	1	[100]

ID	Database	Source title	Figure	Reference
G17	Google Images	ML lifecycle architecture diagram	5	[66]
G23	Google Images	Automatic Recognition of Microstructures of Air-Plasma-Sprayed Thermal Barrier Coatings Using a Deep Convolutional Neural Network	2	[101]
G40	Google Images	Machine Learning in Production: From Models to Systems	6	[102]
G41	Google Images	Integrating Machine Learning with Microscope Control using INTERSECT	1	[103]
G43	Google Images	Dynamic data-driven learning for self-healing avionics	3	[104]
G68	Google Images	A Software System for Robotic Learning by Experimentation	2	[60]
G69	Google Images	An online machine learning framework for early detection of product failures in an Industry 4.0 context	2	[105]
G87	Google Images	Multi-Model Decision-Maker	1	[65]
G106	Google Patents	Online learning and vehicle control method based on reinforcement learning without active exploration	2	[106]
G108	Google Patents	Noise reduction using machine learning	1	[59]
G116	Google Patents	Controlling an autonomous vehicle using cost maps	9	[107]
G140	Google Patents	Adversarial pretraining of machine learning models	3	[108]
G144	Google Patents	Embedded automotive perception with machine learning classification of sensor data	3	[109]
G180	Google Patents	Sentiment and rules-based equity analysis using customized neural networks in multi-layer, machine learning-based model	2	[110]
G195	Google Patents	Neural network-based action detection	2	[111]

Table A.1: Table of the component models found in the multivocal literature review.

B

Data Extraction Form

Source	System eval. and improvement	Runtime eval. and improvement	Parallel independent models	Ensemble learning	Sequentially dependant models	Prediction cache	Post processing	Expert validation	Server-side ML model	Situation-based model selection	Feature extraction	Orchestrator	Data transformation	Multi-layer pattern
W1				•										
W5														
W9														
W10													•	
W24									•			•		
W30			•											
W31		•												
W35			•		•		•			•				
W37											•			
W49													•	•
W53			•											
W55														
W95	•													•
W120									•			•		
W155			•	•	•	•							•	•
W167				•	•			•					•	•
W201														
W217														
W232														
W259	•													•
W380														
W394														
W405									•					
W442	•							•						
W469					•		•			•				
W481			•				•							
W491			•				•							
W500				•							•			
W502					•									
W518														
W537							•						•	

Table B.1: Part of the data extraction form for white literature, showing which patterns were found in which component models.

Source	System eval. and improvement	Runtime eval. and improvement	Parallel independent models	Ensemble learning	Sequentially dependant models	Prediction cache	Post processing	Expert validation	Server-side ML model	Situation-based model selection	Feature extraction	Orchestrator	Data transformation	Multi-layer pattern
G6		●							●		●	●		
G11							●							
G12												●		
G17		●												
G23		●												
G40														
G41														
G43									●			●		
G68						●					●			
G69														
G87				●				●						
G106														
G108							●						●	
G116					●									
G140													●	
G144			●											
G180													●	
G195											●			

Table B.2: Part of the data extraction form for grey literature, showing which patterns were found in which component models.

C

Review Protocol

This document specifies the review protocol for the study The Impact of Design Patterns on Quality Attributes in ML-Enabled Systems and includes the following sections:

- Research question
- Search strategy
- Study selection criteria
- Study selection procedures
- Data extraction strategy

C.1 Research Question

RQ1: What common patterns can be derived from existing component models of ML-enabled software systems in literature?

C.2 Search Strategy

The search will be preformed in the following databases:

White literature: IEEE Xplore, Wiley's, ACM DL, SpringerLink

Grey literature: Google Image Search, Google Patents

The grey literature will be searched using an incognito mode in the browser to avoid search results tailored to our profiles.

The search string will be the following:

```
("component model*" OR "architectural description language*"
OR "component diagram*" OR "class responsibility collaborator*" OR
"structure chart*" OR "component based design*")
AND
("ml-system" OR "ml-enabled system" OR "machine learning" OR
"deep learning" OR "reinforcement learning" OR
"unsupervised learning" OR "supervised learning" OR "neural network")
```

```
AND
NOT "artificial intelligence"
```

For the search on grey literature we will use modified search strings. When searching Google Images, we will use the following string:

```
"machine learning" AND ("component model" OR "component diagram")
```

For Google Patents, since a pattern will most likely include an architectural diagram anyway we do not need to search for it explicitly. Therefore the search string will instead be:

```
"ml-system" OR "ml-enabled system" OR "machine learning" OR
"deep learning" OR "reinforcement learning" OR "unsupervised learning"
OR "supervised learning" OR "neural network"
```

C.3 Study Selection Criteria

Inclusion criteria:

- The system the source mentions is a software system.
- The system the source mentions is a machine learning system.
- The focus of the source is on the system itself.
- The source contains one or more figures showing the relationship between architectural components.

White literature exclusion criteria:

- The source was not written in English.
- The source was a duplicate.
- The source was published before 2014.
- The source was not available in full text.
- The source was not a single chapter or paper.

Grey literature exclusion criteria:

- The source was not written in English.
- The source was a duplicate.
- The source was published before 2014.
- The source was video or audio.
- The source contained component models or equivalent only as an example of what they are or how they are used.

Stopping criteria for grey literature:

- First 100 images of Google Image Search.
- First 100 patents.

C.4 Study Selection Procedures

We will strive for only using sources written in English. However, if it turns out that the amount of component models we can retrieve from English sources is limited, we will relieve this exclusion criteria. In this case, we will prioritise languages that works well with translating.

Each study will be assessed by one person. If some disagreement arises on whether or not a source contains a component model, we will ask Vladislav or Daniel for a third opinion.

C.5 Data Extraction Strategy

When a component model has been identified, we will copy and store it. Additional info on the type of system, as well as what type of ML is used, will also be stored. If there are any disagreements on the extracted models we will ask Vladislav or Daniel for a third opinion.

D

Interview Script

1. We will start by giving you some general information.
    (a) This interview is for our master thesis "Quality Attribute Impact of Design Patterns in ML-Enabled Systems" that we are doing on the Software Engineering master program.
    (b) Our goal with the thesis is to find architectural design patterns in component models and evaluate how these impact the overall system in terms of quality attributes.
    (c) We have done a literature review to find patterns, which we now want to evaluate in terms of their impact on quality attributes, which is where you come in.
    (d) The goal for this interview is to get your expert thoughts and ideas on how certain patterns impact quality attributes of ML-systems. We would first and foremost want to know the impact it has on the system as a whole, but how it affects parts of the system, for example the ML model, is also valuable.
    (e) The interview is planned to take 30 minutes. Do you want us to stop after half an hour or can we finish up the last few questions after that point?
    (f) We encourage you to speak your mind, there are no right and wrong answers.
    (g) Everything you say will be anonymous.
    (h) Is it okay if we record this interview?
        i. Do you have any questions before we start?
2. Could you very briefly share your experience with machine learning and machine learning systems?
    (a) Do you research anything in areas related to ML?
    (b) Have you worked with ML in or in collaboration with industry?
3. [Show quality model]

   (a) Do you want us to go through the quality attributes quickly or are you familiar with them?
   (b) Do you have any questions about the quality model?
4. [Show pattern read aloud definition, explain example]
   (a) Do you understand the pattern?
   (b) Have you used or seen this pattern before?
      i. [If yes:] When? Where? How?
   (c) Which quality attributes do you think this pattern affects and how?
      i. We are not fussed about the minor details, and are looking for direct impacts as a consequence of the pattern so impact depending on specific implementation details is not really relevant.
      ii. You don't have to go through all quality attributes, just share your thoughts on those that stick out the most to you.
   (d) On a scale of -3 to 3, how much does the pattern affect the mentioned QA of the system and why?
      i. Consider that some QA can be both positively and negatively impacted from a pattern.
      ii. In these cases, you can give multiple answers for different reasons or contexts. Don't take an average of them.
   (e) Anything else? [Go back]
   (f) [If no attribute affects system negatively has been mentioned, specifically ask:] Any QAs that are affected negatively?
5. [Go back to 4 until time is up]
6. That was everything for this interview. Thank you so much for participating!
   (a) Would you like us to send you the thesis after it is finished?